新世纪高等学校教材



# 汇编语言程序设计实验指导及习题集

吴向军　李　磊　梁志勇

高等教育出版社
http://www.hep.com.cn
http://www.hep.edu.cn

新世纪高等学校教材

# 汇编语言程序设计实验指导及习题集

吴向军　李磊　梁志勇

高等教育出版社

## 内容提要

本书是与高等教育出版社出版的《汇编语言程序设计》（吴向军等编写）一书相配套的学习辅导书。内容由两部分组成：第一部分是20个实验题，实验内容与教材内容基本上一致，从最基本的调试工具开始，由浅到深、循序渐进来安排实验题，这些实验题在中山大学计算机科学系本科生教学过程中进行了试用，收到了良好的效果。第二部分是教材中大部分练习的参考解答，供学生学习时参考。

在本书的实验部分，还介绍了MASM V6.x编程环境中集成工具——PWB的使用方法，该工具不仅与TC相似，简单易学，而且还有在线帮助，包括：汇编指令的指令格式和功能、ASCII码表、各类按键编码和各种中断的功能等。此外，还介绍了其他编程工具，这些工具允许学生自行设置编程环境，从而使学生真正了解到编程环境的各个方面。

本书可作为高等学校计算机及相关学科的本、专科生学习汇编语言程序设计课程时的辅助资料，也可供应用开发人员参考使用。

**图书在版编目(CIP)数据**

汇编语言程序设计实验指导及习题集/吴向军，李磊，梁志勇. —北京：高等教育出版社，2004.6(2005 重印)

ISBN 7-04-014237-6

Ⅰ.汇… Ⅱ.①吴… ②李… ③梁… Ⅲ.汇编语言-程序设计-高等学校：技术学校-教学参考资料

Ⅳ.TP313

中国版本图书馆 CIP 数据核字（2004）第 042838 号

**策划编辑** 董建波 **责任编辑** 董建波 **封面设计** 王凌波
**版式设计** 胡志萍 **责任校对** 杨雪莲 **责任印制** 孔 源

| | | | |
|---|---|---|---|
| **出版发行** | 高等教育出版社 | **购书热线** | 010-58581118 |
| **社 址** | 北京市西城区德外大街 4 号 | **免费咨询** | 800-810-0598 |
| **邮政编码** | 100011 | **网 址** | http://www.hep.edu.cn |
| **总 机** | 010-58581000 | | http://www.hep.com.cn |
| | | **网上订购** | http://www.landraco.com |
| **经 销** | 北京蓝色畅想图书发行有限公司 | | http://www.landraco.com.cn |
| **印 刷** | 北京明月印务有限责任公司 | | |
| **开 本** | 787×1092 1/16 | **版 次** | 2004 年 6 月第 1 版 |
| **印 张** | 15.5 | **印 次** | 2005 年 11 月第 3 次印刷 |
| **字 数** | 320 000 | **定 价** | 19.50 元 |

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

**版权所有 侵权必究**

**物料号 14237-A0**

# 前　言

汇编语言程序设计是计算机专业中一门重要的基础课程，是培养学生直接使用计算机硬件资源能力的一门课程。它不仅能帮助学生进一步理解计算机组成原理中的各种概念，而且还为其他课程，如：操作系统、接口与通信技术和计算机控制技术等，提供必要的预备知识。

汇编语言是与机器相关的编程语言，用它编写的程序可直接利用 CPU 中的资源，也可直接控制各种端口，或利用中断来控制各种硬件资源，程序的执行效率高。因此，有些系统软件的内核和应用程序的核心部分仍用汇编语言来编写。另外，在工业控制方面，汇编语言的应用也很多，所以，通过实习掌握汇编语言的编程和调试方法对将来的应用有极大的帮助。

本书是与高等教育出版社出版的《汇编语言程序设计》(吴向军等编写)相配套的辅助用书，对该教材中的大部分练习给出了一种参考解答，供学生学习时参考，希望起到“抛砖引玉”的作用。

学习汇编语言的一个重要环节就是上机实验，通过实验能对汇编指令的执行效果有一个更深刻的理解，也能对数据在内存的存储形式和变化有一个更清晰的认识。

在本书的实验部分，我们介绍了 MASM V6.x 编程环境下集成工具——PWB 的使用方法，该工具不仅与 TC 相似，简单易学，而且还有在线帮助，包括：汇编指令的指令格式和功能、ASCII 码表、各类按键编码和各种中断的功能等。此外，还介绍了其他编程工具，这些工具允许学生自行设置编程环境，从而使学生真正了解到编程环境的各个方面。

实验部分包括 20 道实验题，实验内容与教材内容的顺序基本上一致，从最基本的调试工具开始，“由浅到深，循序渐进”来安排实验题。这些实验题在中山大学计算机科学系的本科生教学过程中进行了试用，收到了良好的效果。

编者结合多年的教学实践和汇编语言的特点，建议：实验时间要与课堂讲授时间相当，在条件允许的情况下，二者时间之比也可超过 1:1，部分实验题也可安排学生在课余时间自行完成。

在本书编写安排上，吴向军副教授负责整体内容的构思和组织，李磊工程师和梁志勇助工对练习和实验做了大量细致的工作。本书的最后定稿由吴向军副教授负责。

在本书编写过程中，李宏新副教授审阅了全部内容，并结合其多年的教学实践经验，对实验内容和实验安排提出了宝贵的建议，在此，向他表示衷心的感谢。

由于编者能力所限，加之编程工具日新月异的发展，书中不足和谬误之处在所难免，

恳请广大读者批评指正。

E-Mail: issxjwu@zsu.edu.cn，lilei@163.net，isslzy@zsu.edu.cn。

编 者

2004 年 4 月

# 目　录


**第一篇　实验指导**……1

实验一　调试工具 DEBUG 的使用……1

实验二　汇编编译工具 MASM 的使用……15

实验三　调试工具的使用……25

实验四　其他编程环境的使用……33

实验五　掌握操作数的各种寻址方式……43

实验六　调试汇编源程序和 COM 可执行程序……45

实验七　使用偏移量伪指令和联合数据类型……49

实验八　掌握汇编语言基本程序结构……53

实验九　将数值转换成字符串……55

实验十　数组的使用……57

实验十一　编写子程序……59

实验十二　键盘输入输出处理……62

实验十三　图形模式编程……65

实验十四　鼠标中断处理……67

实验十五　宏的定义和使用……69

实验十六　磁盘和文件操作……70

实验十七　修改中断(*)……73

实验十八　驻留程序 TSR(*)……75

实验十九　综合练习一……77

实验二十　综合练习二……78

**第二篇　习题集**……79

第一章　预备知识……79

第二章　CPU 资源和存储器……81

第三章　操作数的寻址方式……85

第四章　标识符和表达式……87

第五章　微机 CPU 的指令系统……94

第六章　程序的基本结构……104

第七章　子程序和库……116

第八章　输入输出和中断……126

第九章　宏……160

第十章　应用程序的设计……164

第十一章　数值运算协处理器……193

第十二章　汇编语言和 C 语言……205

**参考文献**……239

# 第一篇 实验指导

## 实验一 调试工具 DEBUG 的使用

### 1.1 实验目的

（1）学习如何在 Windows 的命令模式下启动 DEBUG。

（2）掌握 DEBUG 的常用基本命令。

（3）学习如何用 DEBUG 进行跟踪调试。

### 1.2 预备知识

1．进制转换

熟练掌握二进制、八进制、十进制和十六进制的相互转换算法。

2．寄存器

寄存器是 CPU 内部的数据存储资源，是汇编程序员能直接使用的硬件资源之一。寄存器的存取速度比 Cache 还要快。

在 16 位 CPU 中，总共有 4 个 16 位数据寄存器 AX、BX、CX 和 DX，每个 16 位寄存器又可分为 2 个 8 位寄存器（例如 AX 的高 8 位称为 AH，低 8 位称为 AL）；2 个变址寄存器 DI 和 SI；2 个指针寄存器 SP 和 BP；4 个段寄存器 ES、CS、SS 和 DS；1 个标志寄存器 FLAGS；1 个指令指针寄存器 IP。

3．标志位

标志寄存器 FLAGS 的每个位都可以作为标志位。16 位 CPU 使用其中 8 个位表示溢出、中断、进位等状态。每个标志位都有置位和复位两种状态，它们在 DEBUG 中的表示如表 1.1 所示。

表 1.1 DEBUG 中标志位的符号表示

| 标志名称 | 溢出 | 方向 | 中断 | 负号 | 零 | 辅助进位 | 奇偶 | 进位 |
|---|---|---|---|---|---|---|---|---|
| | OF | DF | IF | SF | ZF | AF | PF | CF |
| 置位状态 | OV | DN | EI | NG | ZR | AC | PE | CY |
| 复位状态 | NV | UP | DI | PL | NZ | NA | PO | NC |

## 1.3 DEBUG 命令集

DEBUG 命令及其含义如表 1.2 所示。

表 1.2 DEBUG 命令及其含义

| 命令格式 | | 功能说明 |
|---|---|---|
| A | [地址] | 输入汇编指令 |
| C | 范围 起始地址 | 对由“范围”指定的区域与“起始地址”指定的同大小区域进行比较，显示不相同的单元 |
| D | [范围] | 显示指定范围内的内存单元内容 |
| E | 起始地址 字节值表 | 用值表中的值替换从“起始地址”开始的内存单元内容 |
| F | 范围 字节值表 | 用指定的字节值表来填充内存区域 |
| G | [=起始地址][断点地址] | 从起点（或当前地点）开始执行，到终点结束 |
| H | 数值 1 数值 2 | 显示两个十六进制数值之和、差 |
| I | 端口地址 | 从计算机输入端口读取数据并显示 |
| L | [地址 [驱动器号 扇区 扇区数]] | 从磁盘中将文件或扇区内容读入内存 |
| M | 范围 地址 | 把“范围”内的字节值传送到从“地址”开始的单元 |
| N | 文件标识符 [文件标识符…] | 指定文件名，为读/写文件做准备 |
| O | 端口地址 字节值 | 向计算机输出端口送出数据 |
| P | [=地址][指令数] | 按执行过程，但不进入子程序调用或软中断 |
| Q | | 退出 DEBUG，回到 DOS 状态 |
| R | [寄存器名] | 显示和修改寄存器内容 |
| S | 范围 字节值表 | 在内存区域内搜索指定的字节值表。如果找到，显示起始地址，否则，什么也不显示 |
| T | [=地址][指令数] | 跟踪执行，从起点（或当前地点）执行若干条指令 |
| U | [范围] | 反汇编，显示机器码所对应的汇编指令 |
| W | [地址 [驱动器号 扇区 扇区数]] | 向磁盘写内容，（BX、CX）为写入字节数 |

关于参数的几点说明：

（1）进制：在 DEBUG 中输入或显示的数据都是十六进制形式；

（2）分隔：命令和参数、参数和参数之间要用空格、逗号或制表符等分隔；

（3）地址：用“段值:偏移量”的形式来表示地址，也可用段寄存器来代表“段值”。例如：1000:0、ds:10、es:200、cs:30 等。

① 范围：表示地址范围，它有两种表示方式：“地址 1 地址 2”和“地址 1 长度”。其中：“地址 1”表示起始地址，要用“段值:偏移量”来表示；“地址 2”表示终止地址，只用“偏移量”来表示；“长度”用字母 L 开头的数值来表示。

例如：100:50 100——段值为 100，偏移量从 50 到 100 的内存区域；

100:50 L100——段值为 100，偏移量从 50 开始的 100 个字节区域。

② 端口地址：2 位十六进制数值；

③ 字节值：2 位十六进制数值；

④ 字节值表：由若干个字节值组成，也可以是用引号括起来的字符串；

⑤ 驱动器号：0 表示驱动器 A、1 表示驱动器 B、2 表示驱动器 C、3 表示驱动器 D 等。

## 1.4 DEBUG 命令示例

### 1.4.1 启动 DEBUG

1．打开 Windows 命令窗口

打开命令窗口的步骤：选择“开始”→“运行”。

（1）在 Windows 95/98 环境中，输入“command”命令；

（2）在 Windows 2000 环境中，输入“cmd”命令，如图 1.1 所示。

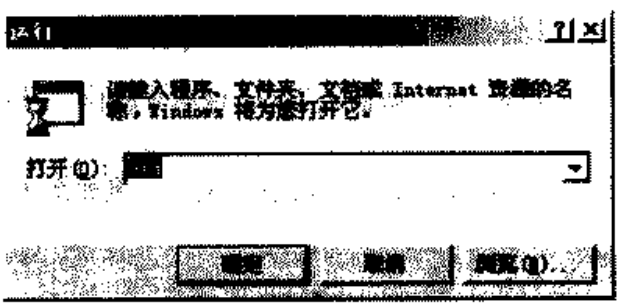



图 1.1 Windows 2000 环境下运行命令窗口

2．启动 DEBUG

在命令窗口中启动 DEBUG，一般命令如下：

DEBUG [文件名][参数表]

其中：文件名指定被调试的文件，其包括名称和后缀，参数表是被调试文件运行时所需要的参数。被调试的文件可以是系统中的任何文件，但通常它们的后缀为.EXE 或.COM。

当 DEBUG 启动成功后，将显示连接符“-”，这时，可输入各种 DEBUG 命令。DEBUG 中标志位的符号表示如表 1.1 所示，其所有命令及其含义如表 1.2 所示。

关于使用命令的几点说明：

（1）在提示符“-”下才能输入命令，在按“回车”键后，该命令才开始执行；

（2）命令是单个字母，命令和参数的大小写可混合输入；

（3）可用 F1、F2、F3、Ins、Del、←、→等编辑键来编辑本行命令；

（4）当命令出现语法错误时，将在出错位置显示“^Error”；

（5）可用 Ctrl+C 或 Ctrl+Break 来终止当前命令的执行，还可用 Ctrl+S 来暂停屏幕显示（当连续不断地显示信息时）；

（6）在 DEBUG 中使用的数都是以十六进制来表示；

（7）DEBUG 中的命令不区分大小写。

下面对 DEBUG 中的命令逐个介绍，并实现相应的示例来加以说明。由于不同计算机环境的差异，下面所给的输出结果在不同计算机中可能有所不同。

### 1.4.2 R 命令

R 命令作用：显示和修改寄存器的值和标志位的状态。

输入命令 R，将显示所有寄存器的值和标志位的状态。执行结果如图 1.2 所示。

```
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B07  ES=0B07  SS=0B07  CS=0B07  IP=0100   NV UP EI PL NZ NA PO NC
0B07:0100 1E            PUSH    DS
-_
```

图 1.2 R 命令的执行结果

输入命令 R CX，在提示符“:”后输入 100。该命令的作用是将寄存器 CX 的值设置为 100。

再输入执行 R 命令，查看修改后的寄存器值。执行结果如图 1.3 所示。

```
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B07  ES=0B07  SS=0B07  CS=0B07  IP=0100   NV UP EI PL NZ NA PO NC
0B07:0100 1E            PUSH    DS
-R CX
CX 0000
:100
-R
AX=0000  BX=0000  CX=0100  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B07  ES=0B07  SS=0B07  CS=0B07  IP=0100   NV UP EI PL NZ NA PO NC
0B07:0100 1E            PUSH    DS
-_
```

图 1.3 R CX 命令的执行结果

### 1.4.3 H 命令

H 命令作用：计算两个十六进制数的和与差。

输入命令：H 10 1，执行结果如图 1.4 所示。

```
-H 10 1
0011  000F
-_
```

图 1.4 H 10 1 命令的执行结果

命令的输出结果有两个数，前者是命令中两个参数之和，后者是两个参数之差。

### 1.4.4　D 命令

D 命令作用：显示指定内存区域的内容。

输入命令 D，执行结果如图 1.5 所示。




图 1.5　D 命令的执行结果

说明：（1）XXXX:YYYY ——前者是内存单元的段地址，后者是内存单元的偏移量；

（2）中间显示区域是内存单元内容（十六进制的形式），每行显示 16 个字节的内容；

（3）右边以“字符”形式显示内存单元值。

命令 D 可带参数，也可省略参数。省略参数时，第一次命令 D 显示的内容是从 DS:100 开始的 128 个字节，再执行不带参数的命令 D 时，则从上次的显示位置往下接着显示。

若执行带参数的命令 D，DEBUG 将显示指定地址范围的内容。带参数的方式有三种。

方式一：D 起始位置

DEBUG 从起始位置开始显示 128 个字节的内容。输入命令 D 1AF5:100，执行结果如图 1.6 所示。

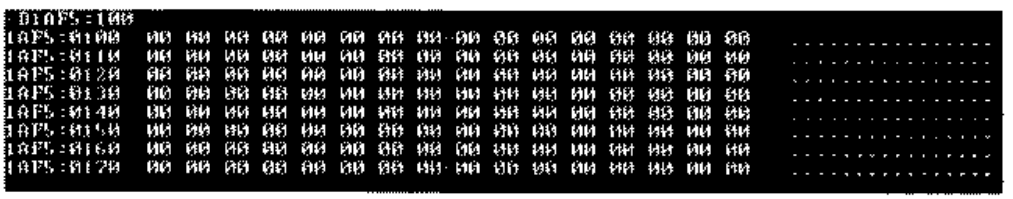



图 1.6　D 1AF5:100 命令的执行结果

方式二：D 起始位置 结束位置。

DEBUG 从起始位置开始一直显示到结束位置。输入命令 D DS:100 11F，执行结果如图 1.7 所示。




图 1.7　D DS:100 11F 命令的执行结果

方式二：D 起始位置 L 长度

DEBUG 从起始位置开始显示指定长度的内容。输入命令 D DS:100 L10，执行结果如图 1.8 所示。




图 1.8 D DS:100 L10 命令的执行结果

### 1.4.5 E 命令

E 命令作用：用字节值表中的值替换从“起始地址”开始的内存单元内容。

E 命令的使用方式：E 起始地址 字节值表

E 命令的使用方式也可以是：E 起始地址。

输入命令：E 1AF5:0100，连续输入 12（空格）、34（空格）、（空格）、56（回车）；再输入命令 D 1AF5:0100，执行结果如图 1.9 所示。

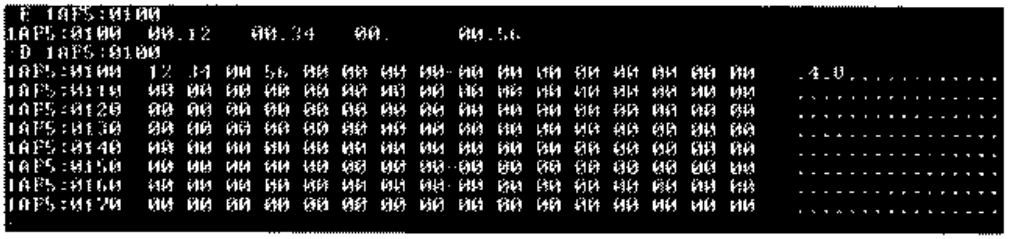



图 1.9 E 1AF5:0100 命令的执行结果

DEBUG 显示 1AF5:0100 单元的内容 12，这时可输入新的数值以达到修改该字节值的目的。如果还要修改后续字节的内容，可按“空格键”继续。当要跳过某个字节时，可以连续按两个空格。

### 1.4.6 F 命令

F 命令作用：用指定的字节值表来填充内存区域。

F 命令的使用方式：F 范围 字节值表

输入命令：F 1AF5:100 L20 1 2 3 4 5，再输入命令 D 1AF5:100，执行结果如图 1.10 所示。

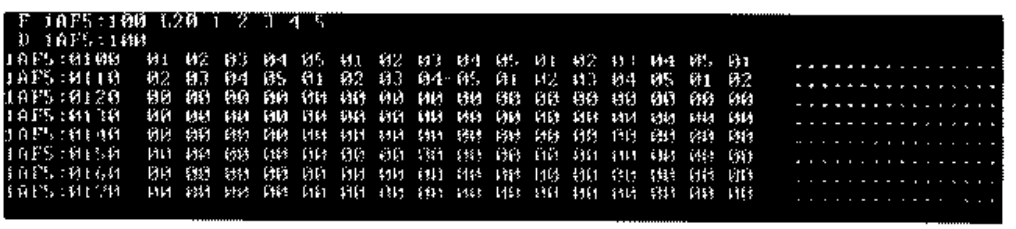



图 1.10 F 1AF5:100 L20 1 2 3 4 5 命令的执行结果

说明：该命令是用数值序列 01、02、03、04、05 依次填充到从 1AF5:100 开始长度为 20H 的内存区域。

输入命令：F 1AF5:100 13F 41 42 43 44，再执行 D 1AF5:100 显示内存单元的内容，执行结果如图 1.11 所示。




图 1.11　F 1AF5:100 13F 41 42 43 44 命令的执行结果

说明：该命令是用数值序列 41、42、43、44 循环填充到 1AF5:[100～13F]的内存区域。

### 1.4.7　M 命令

M 命令作用：把“范围”内的字节值传送到从“地址”开始的单元。

M 命令的使用方式：M 范围 地址

输入命令：M 1AF5:100 13F 1AF5:140，再输入命令 D 1AF5:100 显示内存单元的内容，执行结果如图 1.12 所示。




图 1.12　M 1AF5:100 13F 1AF5:140 命令的执行结果

说明：把内存单元 1AF5:[100～13F]的内容复制到从 1AF5:140 开始的内存单元。

### 1.4.8　C 命令

C 命令作用：对由“范围”指定的区域与“起始地址”指定的同大小区域进行比较，显示不相同的单元。

C 命令的使用方式：C 范围 起始地址

说明：将指定范围的内存区域与从指定地址开始的相同长度的内存区域逐个字节进行比较，列出不同的内容。

输入命令：C 1AF5:100 13F 1AF5:140。由于两块内容完全相同，所以命令执行后没有

任何显示。再输入命令：C 1AF5:100 107 1AF5:180，命令执行后列出比较结果不同的各个字节。命令执行结果如图 1.13 所示。

```
-C 1AF5:100 13F 1AF5:140
-C 1AF5:100 107 1AF5:180
1AF5:0100  41  00  1AF5:0180
1AF5:0101  42  00  1AF5:0181
1AF5:0102  43  00  1AF5:0182
1AF5:0103  44  00  1AF5:0183
1AF5:0104  41  00  1AF5:0184
1AF5:0105  42  00  1AF5:0185
1AF5:0106  43  00  1AF5:0186
1AF5:0107  44  00  1AF5:0187
-
```

图 1.13 C 命令的执行结果

### 1.4.9 S 命令

S 命令作用：在内存区域内搜索指定的字节值表。如果找到，显示起始地址，否则，什么也不显示。

S 命令的使用方式：S 范围 字节值表

输入命令：D 1AF5:100 11F，显示该区域的内存单元内容。

输入命令：S 1AF5:100 11F 41 42 43 44。搜索该区域是否存在字节串 41 42 43 44，并将搜索结果输出。执行结果如图 1.14 所示。

```
-D 1AF5:100 11F
1AF5:0100  41 42 43 44 41 42 43 44-41 42 43 44 41 42 43 44   ABCDABCDABCDABCD
1AF5:0110  41 42 43 44 41 42 43 44 41 42 43 44 41 42 43 44   ABCDABCDABCDABCD
-S 1AF5:100 11F 41 42 43 44
1AF5:0100
1AF5:0104
1AF5:0108
1AF5:010C
1AF5:0110
1AF5:0114
1AF5:0118
1AF5:011C
-
```

图 1.14 S 1AF5:100 11F 41 42 43 44 命令的执行结果

从上面的执行结果可以看出，总共搜索出 8 处匹配。

### 1.4.10 A 命令

A 命令作用：输入汇编指令。

以下的程序要在屏幕上显示“ABCD”四个字符。

先用 E CS:200 命令将“ABCD$”四个字符预先放在内存 CS:200 处（“ABCD$”的 ASCII 码的十六进制数分别为：41、42、43、44、24），然后用命令 D CS:200 20F 显示内存数据内容，最后用命令 A 100 输入以下汇编程序指令：

```
XXXX:0100 MOV AX,CS
XXXX:0102 MOV DS,AX
XXXX:0104 MOV DX,200
XXXX:0107 MOV AH,9
XXXX:0109 INT 21
XXXX:010B INT 20
```

执行结果如图 1.15 所示。

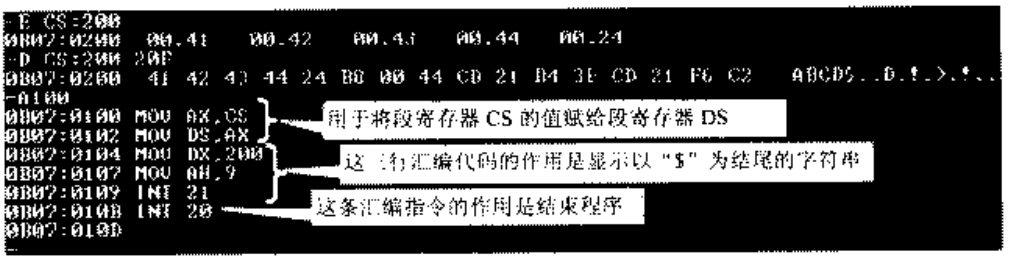



图 1.15　E CS:200 和 A100 命令的执行结果

## 1.4.11　G 命令

G 命令作用：从起点（或当前地点）开始执行汇编指令，到终点结束。

G 命令的使用方式：G [=起始地址] [断点地址]

其作用是从起始地址开始执行到断点地址。如果不设置断点，则程序一直运行到终止指令才停止。

在完成上节的内存数据并输完相应的程序后，在 DEBUG 中执行命令 G=100，输出结果如图 1.16 所示。




图 1.16　G=100 命令的执行结果

汇编程序运行后在屏幕上显示出“ABCD”4 个字符。

在 DEBUG 中，再执行 G =100 10B，其含义是从地址 CS:100 开始，一直运行到 CS:10B 停止。其执行结果如图 1.17 所示。




图 1.17　G=100 10B 命令的执行结果

命令执行后，不但显示出字符串“ABCD”，而且还列出当前寄存器和标志位的值。

## 1.4.12　U 命令

U 命令作用：反汇编，显示机器码所对应的汇编指令。

U 命令的使用方式：U [范围]

如果命令中不写“范围”参数，那么，把从 CS:100 开始的 20H 个字节内容反汇编成汇编指令，并显示出来。若再用命令 U，则依次反汇编其后的内存单元。

如果命令中只有“起始地址”，则把从该“起始地址”开始的 20H 个字节内容反汇编成汇编指令。执行命令 U 100，其执行结果如图 1.18 所示。

```
-U 100
0B07:0100 8CC8          MOV     AX,CS
0B07:0102 8ED8          MOV     DS,AX
0B07:0104 BA0002        MOV     DX,0200
0B07:0107 B409          MOV     AH,09
0B07:0109 CD21          INT     21
0B07:010B CD20          INT     20
0B07:010D 803ED9990000  CMP     BYTE PTR [99D9],00
0B07:0112 7408          JZ      011C
0B07:0114 FE061799      INC     BYTE PTR [9917]
0B07:0118 32C0          XOR     AL,AL
0B07:011A EB06          JMP     0122
0B07:011C 3400          XOR     AL,00
0B07:011E F608          ???     BYTE PTR [BP+SI]
```

图 1.18 U 100 命令的执行结果

如果命令中只有“起始地址”和“终止地址”，则把从“起始地址”到“终止地址”之间的字节内容反汇编成汇编指令。

命令 U 100 10C 的作用是对从 CS:100 到 10B 的内存单元进行反汇编，其执行结果如图 1.19 所示。

```
-U 100 10C
0B07:0100 8CC8          MOV     AX,CS
0B07:0102 8ED8          MOV     DS,AX
0B07:0104 BA0002        MOV     DX,0200
0B07:0107 B409          MOV     AH,09
0B07:0109 CD21          INT     21
0B07:010B CD20          INT     20
```

图 1.19 U 100 10C 命令的执行结果

### 1.4.13 N 命令

N 命令作用：指定文件名，为读/写文件做准备。

以下的 DEBUG 命令序列作用是把刚才的汇编程序保存为 COM 文件。

```
D 200 20F
U 100 10C
N E:\FIRST.COM
R CX
:110
W
```

第一、二条命令的作用是检查一下刚才编写的汇编指令，第三条命令的作用是设置存盘文件名为FIRST.COM，第四条命令的作用是设置存盘文件大小为110H个字节。最后一条命令是将文件存盘。执行结果如图1.20所示。

```
-D 2000 201F
0B07:2000  41 42 43 44 24 00 00 00-CD 21 B4 4F CD 21 B6 C2   ABCD$....!.O.!..
-U 100 10C
0B07:0100 B8C80B          MOV     AX,0BC8
0B07:0103 8ED8            MOV     DS,AX
0B07:0105 BA0000          MOV     DX,0000
0B07:0108 B409            MOV     AH,09
0B07:010A CD21            INT     21
0B07:010C CD20            INT     20
-N E:\FIRST.COM
-R CX
CX 0100
:110
-W
Writing 0110H bytes
```

图1.20　N E:\FIRST.COM命令的执行结果

文件存盘后，在DOS环境下执行FIRST.COM，观看存盘的可执行文件的运行结果，如图1.21所示。

```
E:\>FIRST
ABCD
E:\>
```

图1.21　FIRST.COM文件的运行结果

### 1.4.14　W命令

W命令作用：向磁盘写内容。

在上节“N命令”中可以看出如何使用W命令将文件存盘。

在没有很好地掌握汇编语言和磁盘文件系统前，暂时不要使用W命令写磁盘扇区，否则很容易损坏磁盘文件，甚至破坏整个磁盘的文件系统。

### 1.4.15　L命令

L命令作用：从磁盘中将文件或扇区内容读入内存。

将文件调入内存必须先用DEBUG的N命令设定文件名。下面的命令是将FIRST.COM读入内存。

```
N E:\FIRST.COM
L
```

在文件读入内存后，可用U 100命令查看调入程序的汇编代码。执行结果如图1.22所示。

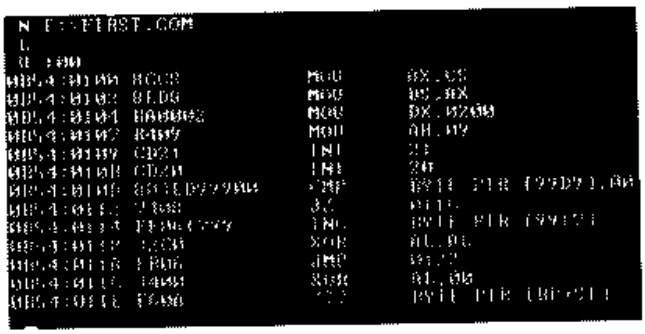


图 1.22 L 命令的执行结果

读取磁盘扇区的方式：L [内存地址[磁盘驱动器号 起始扇区 扇区数]]

内存地址：指定存放读入内容（加载文件或扇区内容）的内存起址，若不指定“内存地址”，则 DEBUG 将使用 CS:100 为当前地址。

磁盘驱动器号：指定包含读取指定扇区的磁盘的驱动器，其值可为：0=A，1=B，2=C 等。

起始扇区：指定要加载其内容的第一个扇区的十六进制数。“扇区数”指定要加载其内容的连续扇区的十六进制数。

只有要加载特定扇区的内容时，才能使用[磁盘驱动器号][起始扇区][扇区数]参数。

例如：要将 C 盘第一扇区读取到内存 DS:300 的位置，相应的 DEBUG 命令为 L DS:300 2 1 1。但是由于 Windows 操作系统对文件系统的保护，这条命令可能会被操作系统禁止运行，如图 1.23 所示。

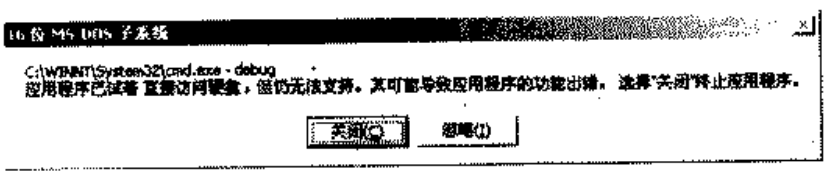



图 1.23 Windows 的错误提示

### 1.4.16 T 命令

T 命令作用：跟踪执行，从起点（或当前点）执行若干条指令。

T 命令的使用方式：T [=地址] [指令数]

若不写“地址”，则 T 命令从 CS：IP 处开始执行，“指令数”是要执行的指令数。若只用 T 命令，则从 CS:IP 处执行一条指令。

执行以下 6 条命令：

```
N E:\FIRST.COM
L
```

```
U 100 10B
R
T=100
T
```

第一、二条命令是装入文件 FIRST.COM，第三条命令是列出程序反汇编代码，第四条命令是显示当前寄存器值，第五条命令是从 CS:100 处开始单步跟踪，第六条命令是继续跟踪后续的指令。执行情况如图 1.24 所示。

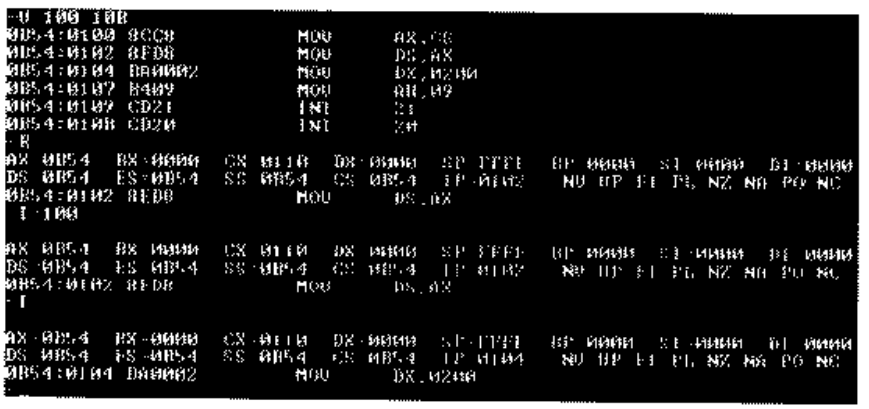



图 1.24　对 FIRST.COM 进行单步跟踪的过程

## 1.4.17　P 命令

P 命令作用：执行汇编程序，单步跟踪。

与 T 命令不同的是：P 命令不会跟踪进入子程序或软中断。其使用方式与 T 命令相同，如图 1.25 所示。

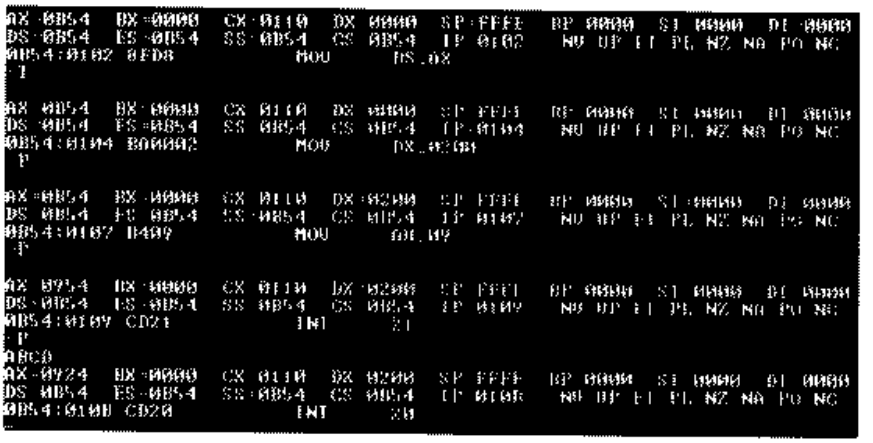



图 1.25　T 命令与 P 命令的比较

### 1.4.18 I 命令

I 命令作用：从计算机输入端口读取数据并显示。
I 命令的用法：I 端口地址
例如从 3F8 号端口读取数据并显示的命令为：I 3F8。

### 1.4.19 O 命令

O 命令作用：向计算机输出端口送出数据。
O 命令的用法：O 端口地址 字节值
例如向 278 号端口发出数据 20H 的命令为：O 278 20。

### 1.4.20 Q 命令

Q 命令的作用是退出 DEBUG，回到 DOS 状态。

# 实验二　汇编编译工具 MASM 的使用

## 2.1　实 验 目 的

（1）学习如何安装 MASM。

（2）学习如何使用 MASM 编译和链接汇编语言程序。

## 2.2　基 础 知 识

1．常用汇编语言工具

常用汇编语言工具有：微软汇编工具 Masm V6.11，微软 Win 32 汇编工具 Masm 32 V6.0，Borland 汇编工具 Tasm V5.0，多功能编辑器 Edit Plus 等。

2．DOS 汇编与 WIN32 汇编

在 DOS 环境下编写汇编程序，程序员可以管理系统的所有资源，访问和修改系统中所有的内存，包括内存控制块、中断向量表、直接对 I/O 端口进行读写。DOS 是个单任务操作系统，只有一个运行级别，任何进程和 DOS 操作系统都是同等级别的。因此，DOS 系统中各个进程会互相影响。如果某个进程死机的话，整个系统都会垮掉。

DOS 工作在实模式下，程序员可以寻址 1MB 的内存，通过段寄存器来指定段的初始地址，每个段的大小为 64KB。超过 1MB 的部分，就只能把它作为 XMS 使用。1MB 以上的内存空间只能存放数据而无法执行程序。

Win32 是指 32 位的 Windows 操作系统，进程有多种运行级别。操作系统工作在最高级别——0 级，而应用程序工作在第 3 级。在第 3 级别，进程无法直接访问 I/O 端口，无法访问其他进程运行的内存，无法向自己的运行代码写入数据。只有对于最高级别的进程，系统才是全开放的。

Windows 工作在保护模式下，所有的资源对进程来说都是被“保护”的。在内存方面，Windows 使用了处理器的分页机制，程序员不必用一个段寄存器去指定段的地址，因为在保护模式下，段寄存器的含义是不同的（参见有关 80386 方面的书籍）。程序员可以直接指定一个 32 位的地址来寻址 4GB 的内存。在程序结构方面，Windows 程序也有很大的不同，它是“基于消息”的。

在上面列举的汇编工具中，一些是 DOS 汇编工具，一些是 Win32 汇编工具。初学者可以从 DOS 汇编入手，先掌握汇编编程的思路、基本语法和编程技巧，待入门后再学习

Win32 汇编。

学习 DOS 汇编可以在 Windows 环境中安装 MASM 6.x 或其他 16 位汇编工具。在命令提示符状态下编译和链接 DOS 汇编程序。DOS 汇编程序在 Win32 环境中运行时，操作系统会模拟实模式运行 16 位进程。对于一些可能会影响整个操作系统工作的指令，Win32 操作系统会拒绝执行。

## 2.3 MASM 的安装

在“命令提示符”下运行 Setup.exe 程序，开始安装 MASM。安装 MASM 的主窗口如图 2.1 所示。

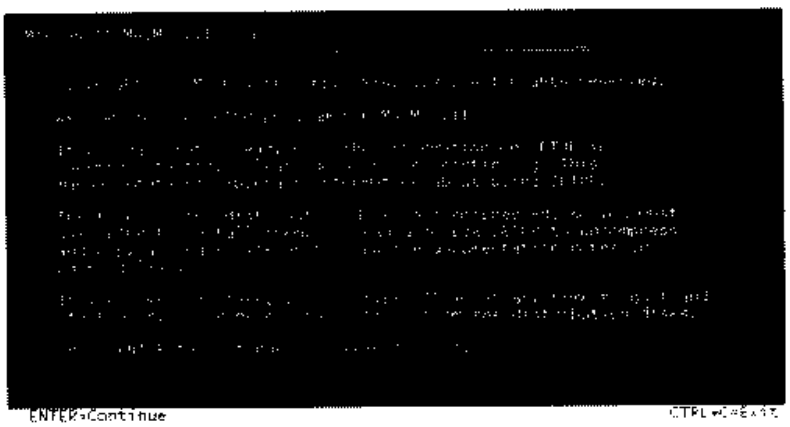


图 2.1 安装 MASM 的主窗口

安装主菜单如图 2.2 所示，在其上选择“Install the Microsoft Macro Assembler”，再继续下述安装步骤。

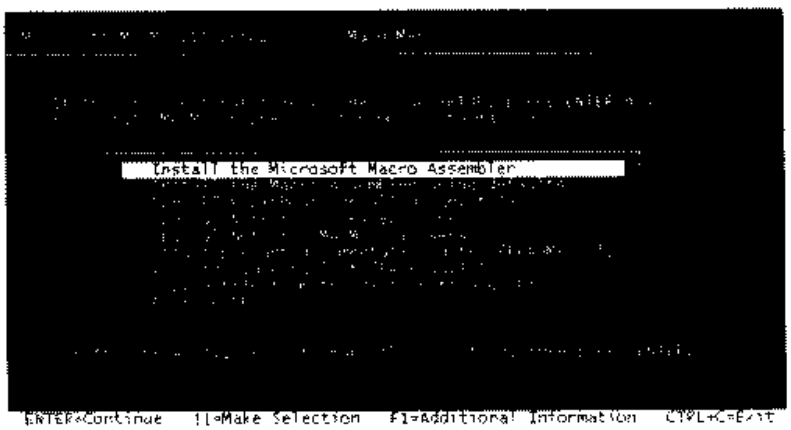



图 2.2 安装主菜单

在以后的显示窗口中，依次选择 DOS/Windows & NT、Windows 文件、PWB（Programmer's WorkBench）、Brief compatibility（可不安装）、Microsoft Mouse Driver、MASM.exe、帮助文件（可选）、Sample Program（编程的例子，可选）等。

以上操作是选择安装内容，图 2.3 是选择安装到计算机硬盘的哪一个分区。安装程序将根据前面选择的安装内容来显示相应的窗口。




图 2.3　选择系统安装的硬盘分区

选择系统的安装目录，例如将系统安装在 E 盘，如图 2.3 所示，然后是询问各安装内容的存放目录，一般情况下最好选择由系统自行安装的目录。

以上操作完成后，安装程序将列出前面所做的所有安装设置，要求进行确认，如图 2.4 所示。




图 2.4　列出前面所做的所有安装设置

若无需改动安装选项，则选择“NO CHANGES”开始安装。

文件复制完毕后，安装程序将会提示 MASM 6.11 与 MASM 5.1 版本的区别，具体的

内容可以查看安装菜单中的“Special Note for MASM v5.1 users”或者编程指南中的附录 A。

安装程序还给出“环境设置”，提示用户如何修改计算机系统中有关配置文件，这些窗口的显示内容是用户很容易忽略的。如果不做相应的修改，将会导致 MASM 的使用不便。

具体的环境设置窗口如图 2.5 所示。

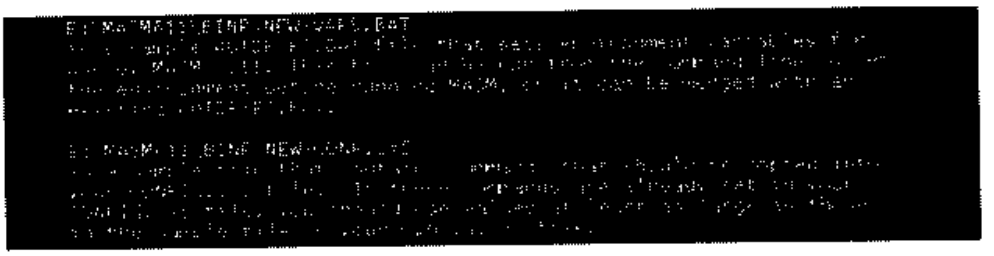


图 2.5 MASM 的环境配置文件

用 NEW-VARS.BAT 和 NEW-CONF.SYS 两个文件的内容分别修改计算机系统中的 AUTOEXEC.BAT 和 CONFIG.SYS 文件，这些环境参数改变后需重新启动计算机才能生效。接下来列举 Windows 环境中的配置样例文件，如图 2.6 所示。

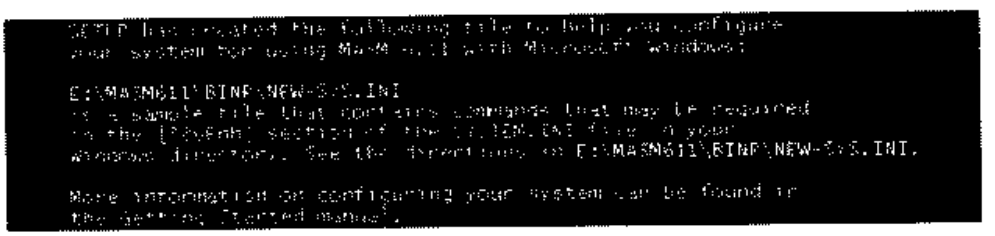


图 2.6 MASM 的 Windows 环境配置样例文件

在现在的 Windows 系统环境下，可以不用 NEW-SYS.INI 的内容去修改 SYSTEM.INI。

下面是 PWB 环境的设置，对 TOOLS.PRE 改名或把其内容放入到 TOOLS.INI 文件中，如图 2.7 所示。

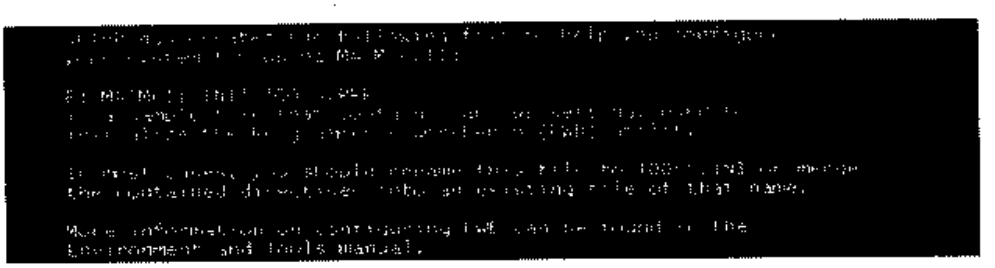


图 2.7 PWB 的环境配置文件

最后是有关内存方面的设置，在现在的 Windows 环境下，可不理会这些修改。

在安装结束前最好看一看“Special Note for CodeView users”、“Special Note for MASM v5.1 users”、“README.TXT”和“PACKING.TXT”的内容，然后退出安装，如图 2.8 所示。

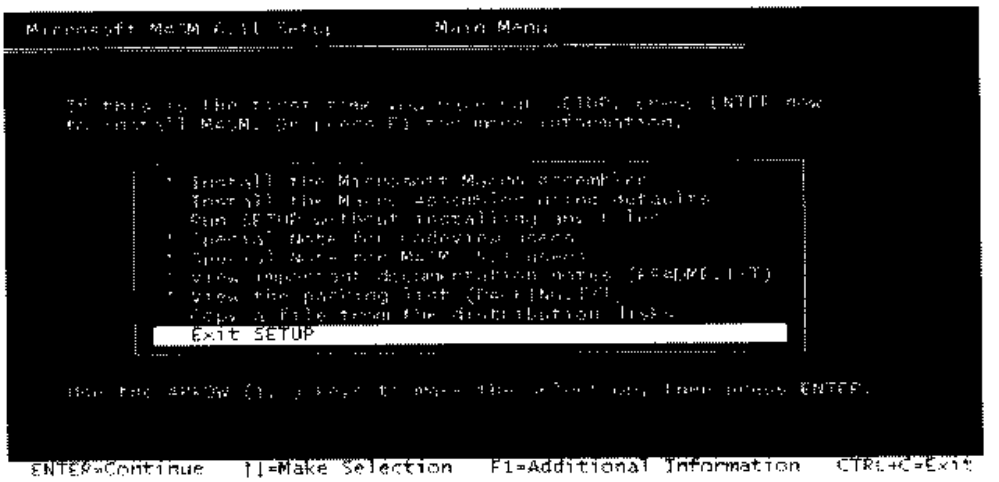



图 2.8　退出安装

## 2.4　MASM 的使用

MASM.EXE 用于将汇编源程序汇编成目标文件。当 MASM 汇编环境参数配置好之后，在“命令提示符”窗口中键入“MASM /H”指令，将显示出 MASM 的使用介绍。

MASM 的命令格式如下：

MASM [option...] source(.asm),[out(.obj)],[list(.lst)],[cref(.crf)][;]

（1）source(.asm)　汇编源文件的名称，默认的扩展名是 ASM。

（2）out(.obj)　输出的目标代码文件，默认的文件名与汇编源文件相同。

（3）list(.lst)　输出的列表文件，默认情况下，MASM 不生成列表文件。

（4）cref(.crf)　输出的交叉参考文件，默认情况下，MASM 不生成交叉参考文件。

（5）最后的分号表示其后的选项取默认的设置。

MASM 中最重要的几个选项说明，如表 2.1 所示。

表 2.1　MASM 中最重要的几个选项说明

| 选　项 | 选项的含义 | 选　项 | 选项的含义 |
|---|---|---|---|
| /H、/HELP | 显示 MASM 的使用帮助 | /I<path> | 寻找引入文件（INC）的目录 |
| /L | 生成列表文件 | /La | 生成全部列表文件 |
| /ML | 区分所有标识符号的大小写 | /MU | 将全局标识符号转换为大写 |

续表

| 选 项 | 选项的含义 | 选 项 | 选项的含义 |
|---|---|---|---|
| /MX | 区分全局标识符号的大小写 | /N | 隐藏列表文件中的符号表 |
| /T | 隐藏成功汇编时的显示消息 | /W[0/1/2] | 设置告警等级（0：无；1：严重；2：警告） |
| /Zi | 生成 CodeView 的行号信息 | /Zd | 生成 CodeView 的标识符号信息 |

下面用一个汇编程序说明 MASM.EXE 的使用方法。

```
data1   segment
        msg db  "Hello, world.",0dh,0ah,"$"
data1   ends
code1   segment
        assume  cs:code1, ds:data1
start:
        mov ax, data1
        mov ds, ax
        lea dx, msg
        mov ah, 9
        int 21h
        mov ax, 4c00h
        int 21h
code1   ends
        end start
```

用编辑器输入上面源程序，并保存为 HELLO.ASM，然后用下面命令来生成目标文件（“下划线”部分是用户输入的命令）。

```
…>masm hello;
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.
 Invoking:ML.EXE /I. /Zm /c hello.asm
Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993.  All rights reserved.
Assembling:hello.asm
```

用命令 MASM /La /Zi /Zd HELLO.ASM 可生成 LST 文件，可用 type 或 edit 来阅读生成的 HELLO.LST 的内容。

## 2.5 LINK 的使用

LINK.EXE 用于将目标文件（obj 文件）连接成可执行文件。

LINK 的命令格式如下：

```
LINK [option...] <objs>,<exefile>,<mapfile>,<libs>,<deffile>[;]
```

（1）objs(.obj)　指定目标文件的名称。可以有多个目标文件，目标文件之间用加号或者空格间隔。

（2）exefile　输出可执行文件的名称。默认的文件名与目标文件相同，默认扩展名改为 EXE。

（3）mapfile　指定输出的列表文件的名称，默认的扩展名是 MAP。默认情况下不生成列表文件。

（4）libs　指定链接时使用的库文件，默认的扩展名是 LIB。可以有多个库文件，库文件之间用加号或者空格间隔。默认情况下 MASM 不使用库文件。

（5）deffile　指定输出的定义文件的名称，默认的扩展名是 DEF。默认情况下不生成定义文件。

（6）最后的分号表示其后的项目按默认情况处理。

LINK 中最重要的几个选项说明，如表 2.2 所示。

表 2.2　LINK 中最重要的几个选项说明

| 参数名 | 缩写 | 作　用 |
|---|---|---|
| /Codeview | /Co | 加入 Codeview 的标识符号及列号，该选项与/Exepack 不兼容 |
| /Exepack | /E | 压缩可执行文件的大小，与/Co 和/Inc 不兼容 |
| /Farcalltranslation | /F | 优化远程调用，当使用/Tiny 时自动使用/Farcall。当链接 Windows 程序时不建议同时使用/Farcall 和/Farcalltranslation |
| /Incremental | /Inc | 准备加入 ilink 链接，与/Exepack 和/Tiny 不兼容 |
| /Linenumbers | /Li | 将源文件行号和相关地址加入 MAP 文件，目标文件必须带行号汇编 |
| /Map | /M | 将公共标识符加到 MAP 文件 |
| /Stack:Number | /ST:NNN | 设置栈段的最大字节数，不超过 64KB |
| /Tiny | /T | 生成小模式的 COM 文件，与/Incr 不兼容 |
| /?、/Help | /?、/He | 显示简要帮助信息 |

下面将刚才用 MASM.EXE 汇编出来的目标程序进行链接。目标程序名称为 HELLO.OBJ。

```
…>link hello;
Microsoft(R)Segmented Executable Linker  Version 5.31.009 Jul 13 1992
Copyright(C)Microsoft Corp 1984-1992.  All rights reserved.
LINK :warning L4021:no stack segment
```

连接警告没有定义栈段，但这不影响程序的运行。

```
…>hello                    //查看该程序的运行结果
```

## 2.6 ML 的使用

ML.EXE 的功能相当于 MASM.EXE 和 LINK.EXE 的总和，可以将汇编语言源程序汇编和链接后直接生成可执行文件。

ML 的命令格式如下：

```
ML [ /options ] filelist [ /link linkoptions ]
```

（1）/options　ML 的各种链接参数。

（2）/link linkoptions　设定的是链接参数，其与上节中的参数相一致。

（3）filelist　指定汇编源文件的名称，默认的扩展名是 ASM。可以有多个汇编源文件，文件之间用空格间隔。

ML 中最重要的几个选项说明，如表 2.3 所示

表 2.3　ML 中最重要的几个选项说明

| 参 数 名 | 参 数 含 义 | 参 数 名 | 参 数 含 义 |
|---|---|---|---|
| /AT | 允许小模式（生成 COM 文件） | /c | 仅汇编不链接 |
| /Cp | 区分用户标识符的大小写 | /Cu | 将所有标识符映像为大写 |
| /Cx | 区分公共和外部标识符的大小写 | /F <hex> | 设置堆栈大小 |
| /Fe<file> | 设置可执行文件的名称 | /Fl[file] | 生成汇编代码列表文件 |
| /Fo<file> | 设置目标文件的名称 | /FPi | 生成 80x87 模拟代码 |
| /I<name> | 添加 include 文件的目录 | /link <link options & libraries> | 链接选项和库 |
| /Sn | 隐藏列表文件中的符号表 | /Sx | 列表文件中列举错误条件 |
| /WX | 传回警告的错误码 | /W<number> | 设置告警级别（1、2、3） |
| /Zd | 在目标文件中产生 CodeView 行号 | /Zi | 在目标文件中产生 CodeView 标识符 |

下面用 ML.EXE 对 hello.asm 文件进行汇编和链接。

```
...>ML hello
Microsoft(R)Macro Assembler Version 6.00
Copyright(C)Microsoft Corp 1981-1991.  All rights reserved.
 Assembling:hello.asm
Microsoft(R)Segmented-Executable Linker  Version 5.13
Copyright(C)Microsoft Corp 1984-1991.  All rights reserved.
Object Modules [.OBJ]:hello.obj
Run File [temp.exe]:hello.exe
List File [NUL.MAP]:
Libraries [.LIB]:
```

```
Definitions File [NUL.DEF]:
LINK :warning L4021:no stack segment
…>hello                    // 查看该程序的运行结果
```

## 2.7　调试执行文件

用 MASM 编译源文件 HELLO.ASM 之后，用 DEBUG 调试可执行程序 HELLO.EXE。具体的做法：

（1）在“命令提示符”窗口下执行“DEBUG <文件所在路径+文件名>”指令。

（2）执行“R”指令查看段寄存器 CS 和 DS 的值，在 EXE 文件中两值应该不同。

（3）执行“U”指令查看 CS：IP 所指地址的反汇编代码。

（4）程序刚开始时会设置 DS 段寄存器的值，将其指向 DATA1 数据段。用跟踪指令“P”检查这两条指令的执行结果。

（5）DS 段寄存器的值被修改后，执行指令“D DS:0”，查看数据段的具体内容。

（6）继续用指令 P 进行程序跟踪，检查各寄存器和标志位的修改内容以及程序的运行情况。

（7）程序正常结束后，用“R IP”指令修改寄存器 IP 的值，将其设置为 0。

（8）重复第（4）步到第（6）步，改用指令 T 而不是指令 P 对程序进行跟踪，查看跟踪情况。

## 2.8　生成 COM 文件

对于有多个数据段、栈段的汇编语言程序，建议将其汇编和链接成 EXE 文件。如果程序的代码和数据量比较小，可以将其汇编和链接成 COM 文件。

EXE 文件可以有多个段，每个段的最大空间为 64KB。程序执行时需要对多个段进行管理，所以 EXE 文件的结构有些复杂。而 COM 文件只有一个段，执行代码和数据都处于一个段中，其程序的大小不超过 64KB，因此 COM 文件比 EXE 文件显得更加短小精悍。

把前面的源程序 HELLO.ASM 改写成下面形式，并另存为 hello1.asm。

```
code1   segment
        assume cs:code1, ds:code1
        org 100h
start:
        jmp next
        msg db  "Hello, world.",0dh,0ah,"$"
next:
```

```
        mov ax, cs
        mov ds, ax
        mov dx, offset msg
        add dx, 100h
        mov ah, 9
        int 21h
        mov ax, 4c00h
        int 21h
code1   ends
        end start
```

执行下列命令将源文件 hello1.asm 汇编、链接成 hello1.com 文件。

```
...>masm hello1.asm;        // 对源文件进行汇编
...>link /tiny hello1;      // 将目标文件链接成 COM 文件
...>hello1                  // 查看运行结果
```

比较 HELLO.EXE 和 hello1.com 的大小，用 DEBUG 调试程序 hello1.com，并了解它的程序结构。

# 实验三　调试工具的使用

## 3.1　实 验 目 的

（1）学习使用 CV（CodeView）调试程序。

（2）学习使用 PWB（Programmer's WorkBench）编写和调试程序。

（3）学习使用 Turbo Debugger 调试程序。

## 3.2　预 备 知 识

1．什么是 CV

CV 的全称是 CodeView，它是 Microsoft 公司多种软件包（例如 MASM、Microsoft C）里的调试工具，使用 CV 必须先安装这些软件包并且正确设置环境参数。

2．什么是 PWB

PWB 的全称是 Programmer's WorkBench，它是 Microsoft 公司 MASM 软件包里的汇编语言编辑和调试工具，使用 PWB 必须先安装 MASM 软件包并且正确设置环境参数。

3．什么是 Turbo Debugger

Turbo Debugger 是 Borland 公司多种编程软件包（例如 Borland C++、TASM）里的调试工具，使用 Turbo Debuger 最好先安装这些编程软件包并且设置好相应的环境参数。

## 3.3　CV 的 使 用

CV 是一个简单、直观的全屏幕调试工具，它可调试多种语言的源程序所生成的可执行代码。它的作用类似于 DEBUG，但比 DEBUG 的功能更为强大，而且界面也很直观。

CV 的使用方法是：在命令提示符窗口下，执行“CV <被调试可执行文件名称>”指令。CV 的界面如图 3.1 所示。

### 3.3.1　CV 的菜单

CV 主要菜单及其功能的说明如下：

（1）File 菜单包含如下命令：

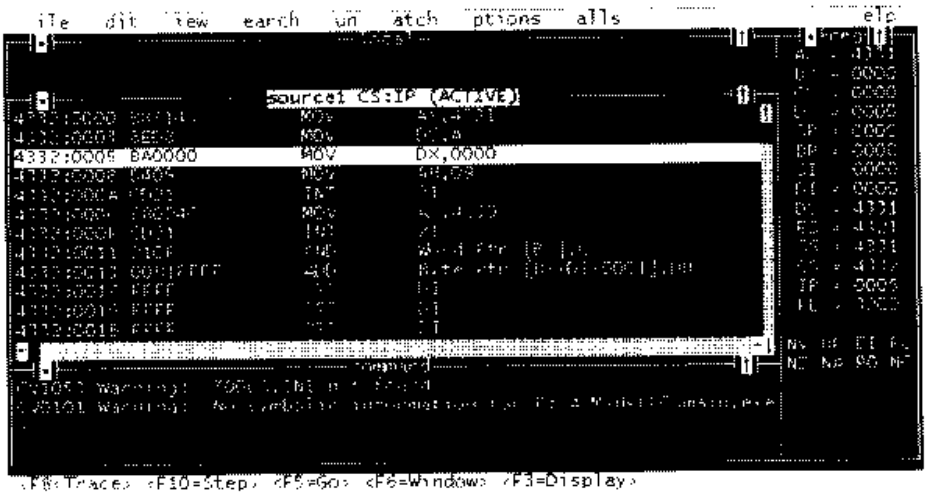



图 3.1 CV 的界面

- Open source... 打开源文件装入到当前活动的资源窗口中；
- Open Module 打开模块文件装入当前活动的资源窗口中；
- Print... 将当前窗口的部分或全部内容写到某个文件中；
- Dos Shell 暂时离开 Codeview 界面，切换到操作系统命令提示符中；
- Exit 退出 CodeView。

（2）Edit 菜单包含如下命令：

- Undo 撤销最后一次编辑动作；
- Copy 将选中的文本存入 CodeView 剪贴板中；
- Paste 将 CodeView 剪贴板的内容粘贴到当前光标位置。

（3）Search 菜单包含如下命令：

- Find... 在源程序窗口中搜索字符串；
- Selected Text 选定文本；
- Repeat Last Find 重复最后一次的搜索；
- Label/Function... 在源程序中搜索标号或功能。

（4）Run 菜单包含如下命令：

- Restart 重新开始；
- Animate 自动连续执行指令；
- Set Runtime Arguments... 设置运行参数；
- History on 记录/停止记录当前调试动作；
- Undo 撤销最后一次调试动作；
- Replay 恢复刚撤销的调试动作。

（5）View 菜单包含如下命令：

- Source　打开反汇编代码窗口，显示内容 CS:IP；
- Memory　打开内存数据窗口，显示内容 DS:0；
- Register　打开/关闭寄存器窗口；
- 8087　打开/关闭 8087 协处理器窗口；
- Watch　打开/关闭监视窗口；
- Command　打开/关闭命令窗口；
- Help　打开/关闭帮助窗口；
- Output　打开/关闭输出窗口。

（6）Watch 菜单包含如下命令：

- Add watch...　增加检查点；
- Delete watch...　删除检查点；
- Set Breakpoint...　设置程序断点；
- Edit Breakpoints...　编辑程序断点；
- Quick watch...　显示当前光标位置变量的值或选中的表达式的值。

（7）Options 菜单包含如下命令：

- Source window...　设置反汇编代码窗口属性；
- Memory Window...　设置内存窗口属性；
- Trace Speed...　设置跟踪速度；
- Language...　设置语言属性。
- Screen Swap
- Replay...　设置再运行输入数据参数；
- Case Sensitivity　设置大小写有关参数；
- 386 Instructions　80386 指令参数；

（8）Calls 菜单　显示当前调试的程序中那些函数被调用。

### 3.3.2　CV 的快捷键

CV 的主要快捷键如表 3.1 所示。

表 3.1　CV 的主要快捷键

| 快　捷　键 | 名　　称 | 作　　用 |
|---|---|---|
| F1 | Help | 帮助 |
| F2 | Register | 显示/隐含寄存器组窗口 |
| F3 | Display | 以不同的显示方式显示当前执行的程序 |
| F4 | Output | 显示程序的输出屏幕 |

续表

| 快 捷 键 | 名 称 | 作 用 |
|---|---|---|
| F5 | Go | 程序从 CS:IP 处开始运行，一直执行到下一个逻辑断点或程序尾 |
| F6 | Window | 依次进入当前屏幕所显示的窗口 |
| F8 | Trace | 单步执行指令，类似于 DEBUG 中的 T 指令 |
| F9 | Set Breakpoint... | 在源程序行中设置/取消断点，用鼠标左键双击断点处也可 |
| F10 | Step | 单步跟踪，但不进入被调用的子程序，类似于 DEBUG 中的 P 指令 |
| Esc | Cancel | 终止当前操作 |
| Alt+F4 | Close CV | 关闭 CodeView |
| Shift+F9 | Quick Watch... | 显示当前光标位置变量的值或选中的表达式的值 |
| Ctrl+U | Delete Watch... | 删除检查点 |
| Ctrl+W | Add Watch... | 增加检查点 |
| Ctrl+F4 | Close | 关闭当前窗口 |

编写一个汇编语言程序 HELLO.ASM，该程序的源代码见实验二。

```
...>ML HELLO.ASM          // 生成执行文件 HELLO.EXE
...>CV HELLO.EXE          // 用 CODEVIEW 调试程序 HELLO.EXE
```

## 3.4 PWB 的使用

PWB 是 MASM 6.11 提供的编程集成环境。在此环境下，程序员可直接编写源程序、汇编、连接和运行。

（1）在汇编源程序时，如果有错误，系统将列举出所有出错位置和出错原因。还可用 Shift+F3 和 Shift+F4 来定位前一个错误和后一个错误。

（2）在连接时，可指定堆栈的大小、附加的库文件、符号跟踪等选项。

（3）在运行时，可设置命令行参数、直接运行、按调试方式运行、用 DOS 命令来运行等。

```
...>pwb /?
.....
Usage:PWB [<options>] [<files>]
    [<options>]
        [/D[S|T|A]...]      Disable CURRENT.STS, TOOLS.INI, Autoload
        [/P[P|F] <file>]    Set PWB or Foreign Program List
        [/PL]               Set Last Program List
        [/E <cmd-string>]   Execute commands
        [/M <mark>]         Start at mark
        [/R]                Read-only mode
    [<files>]
```

```
[/T] <file>...    (Temporary) file to open
```

通常情况下，在 PWB 后面跟一个将要编辑的源文件名。假如要编辑源文件 hello.asm，那么，可直接输入下面命令：

```
...>pwb hello.asm
```

PWB 的界面如图 3.2 所示。

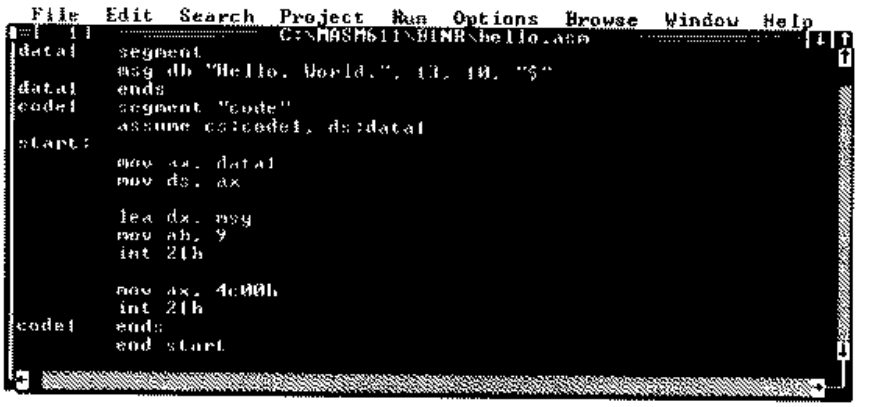



图 3.2　PWB 的界面

1．编辑源文件

PWB 的编辑功能与许多编辑器的功能类似，有建立新文件、保存文件、另存为、光标移动功能、块操作、插入/删除操作、恢复操作、查找/替换操作、设置编辑器的功能键和各类颜色等。要想了解更全面的编辑功能，可查看菜单 File、Edit、Search 和 Options 的菜单项。

2、汇编和连接文件

在集成环境下，源程序的汇编和连接是一次性完成的。当汇编任务结束，并且没有产生错误信息时，连接程序立即开始连接工作。但如果源文件有错，则显示所有错误位置和原因，连接程序不会被执行。在浏览错误信息时，可用 Shift+F3 和 Shift+F4 来定位前一个错误和后一个错误。

3、运行程序

在运行程序时，可设置命令行参数、直接运行、按调试方式运行、用 DOS 命令来运行等。通常情况下，在编写程序的初期，一般都用“调试方式”来运行程序。当选用这种方式时，系统会自动进入 CV 的调试环境。

## 3.5　Turbo Debugger 的使用

Turbo Debugger 系统是 Borland 公司多种编程环境的组成部分，它可以在安装 Borland

公司的 Borland C++等编程环境时默认安装，也有独立的安装程序。

TASM 是 Borland 公司汇编语言编程工具，其主要的执行文件有 TASM、TLINK 和 TD（Turbo Debugger）。

安装完 Turbo Debugger 后需在“我的电脑”→“属性”→“高级”→“环境参数”中设置 Turbo Debugger 的目录路径。如果 Turbo Debugger 安装在“c:\tasm”目录，则在 path 变量中增加以下内容“;c:\tasm\bin;c:\tasm\include”。

如果 Turbo Debugger 是在安装 Borland C++等编程环境时默认安装的，也需要设置环境参数。如果 Borland C++安装在 c:\bc，则需在“我的电脑”→“属性”→“高级”→“环境参数”中设置 Borland C++的目录路径，在 path 变量中增加以下内容：“;c:\bc\bin;c:\bc\include”。

Turbo Debugger 的常用方法：

...>TD 执行文件名称

Turbo Debugger 的界面如图 3.3 所示。

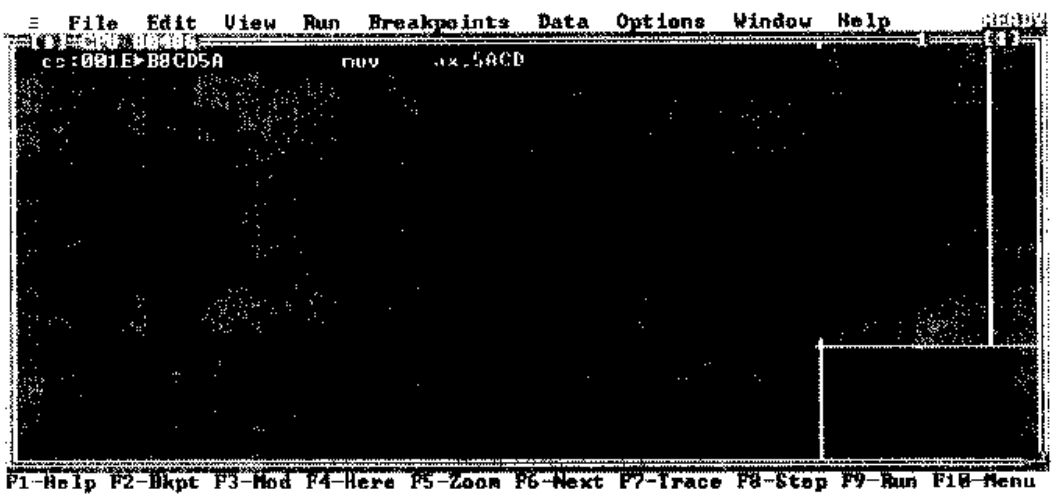



图 3.3 Turbo Debugger 的界面

Turbo Debugger 的主要菜单及其功能如下：

（1）File 菜单包含如下命令：

- Open 装入准备调试的可执行文件；
- Codeguard info... 显示和设置代码保护信息；
- Get Info... 显示被调试文件的有关信息；
- OS shell 运行 DOS 命令进程；
- Quit 退出 Turbo Debugger。

（2）Edit 菜单包含如下命令：

- Copy　将选中的文本存入 Turbo Debugger 剪贴板中；
- Paste　将 Turbo Debugger 剪贴板的内容粘贴到当前光标位置。

（3）View 菜单包含如下命令：

- Breakpoints　设置断点；
- Stack　打开栈段窗口；
- Watchs　打开监视窗口；
- Module...　打开模块窗口；
- Dump　打开数据窗口；
- Registers　打开寄存器窗口；
- Numeric processor　打开数字协处理器窗口；
- Windows messages　打开 Windows 消息窗口；
- Clipboard　打开剪贴板窗口。

（4）Run 菜单包含如下命令：

- Run　以正常速度运行程序；
- Go to cursor　运行到当前光标位置；
- Trace into　单步执行（进入调用内部，类似于 debug 中的 T 指令）；
- Step over　单步执行（不进入调用内部，类似于 debug 中的 P 指令）；
- Execute to...　执行到指定位置；
- Until return　执行到过程调用结束；
- Animate...　自动连续执行指令；
- Program reset　重新装入程序。

（5）Breakpoints 菜单包含如下命令：

- Toggle　在当前光标位置设置断点；
- At　在指定位置设置断点；
- Express true global...　在表达式为真的位置设置断点；
- Delete all　删除所有断点。

（6）Data 菜单包含如下命令：

- Inspect...　在检查窗口中增加一个变量或表达式；
- Evaluate/modify...计算和显示变量或表达式的值，并输入新的值进行更改；
- Add watch...　在监视窗口中增加一个新的表达式。

（7）Options 菜单包含如下命令：

- Display options...　显示选项；
- Set restart options...　设置调试器的重新开始参数。

如果要调试 hello.exe，可用下面的命令：

```
...>TD hello.exe
```

用 Turbo C/C++、Turbo Pascal 或者 Borland C++编写一个 C 或者 Pascal 程序，并生成相应的执行文件，然后试用 Turbo Debugger 对准备好的可执行文件进行调试。

# 实验四　其他编程环境的使用

## 4.1　实 验 目 的

（1）学习使用 Visual Assembly 汇编语言集成编译器。
（2）学习使用 MasmEdit 汇编语言集成编译器。
（3）学习使用 EditPlus 编写汇编语言源程序。
（4）介绍汇编语言编程语法使用辅助工具 HelpPc。

## 4.2　预 备 知 识

1．源程序的编辑、编译、运行和调试

汇编语言是面向机器的低级语言，与高级语言相比，它不需要庞大的库和辅助文件，汇编语言的编辑、编译和调试环境也十分精炼。一般情况下，程序员可以使用 DOS 环境下的 EDIT，Windows 环境下的记事本等来编辑汇编语言源文件，然后用 masm 或 tasm 汇编源程序，最后运行可执行文件，或用 Debug、CodeView 或 Turbo Debugger 等工具来调试程序。

前面所列举的各种汇编语言编程工具虽然简单方便，但其编程环境还不如现在各种高级语言的编程优秀。为了更好地帮助学习和使用汇编语言，提高编程的效率和质量，本次实验介绍一些 Windows 环境下的汇编语言集成编程环境。

2．DEBUG 的使用

DEBUG 是 DOS 和 Windows 操作系统提供的汇编语言调试工具。

在实验一中详细说明了 DEBUG 的使用方法。

3．MASM、TASM 的使用

MASM 和 TASM 是 DOS 和 Windows 环境下提供的汇编语言编译工具。

在实验二中详细说明了 MASM 的使用方法。

4．工具的下载地址

见附录。

## 4.3　Visual Assembly 汇编语言集成编译器

Visual Assembly 是由国内程序员编写的自由软件。该软件不仅是编辑、汇编、运行或

调试汇编语言程序的集成环境，而且还提供了简单的汇编语言源程序的结构模板。此外，软件还带有两个小工具：ASCII 码的查询工具和十进制数/十六进制数的转换工具。

该软件的界面如图 4.1 所示。

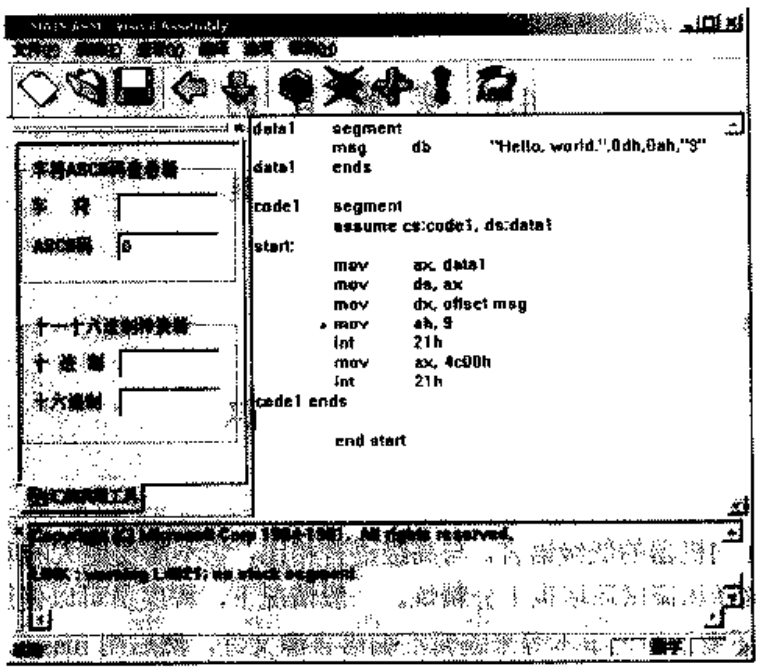



图 4.1 Visual Assembly 的界面

Visual Assembly 本身并不带编译器和调试器，它的编译和调试功能是外挂的。目前支持 MASM、TASM、MCS51 三种编译器，调试器采用的是 DEBUG。

Visual Assembly 第一次使用时，必须在选项菜单中设置编译器的类型及其所在目录。如果系统中已经安装有 MASM 或 TASM 并且设置好工作环境，那么只需要简单地在其选项中指明路径即可。如果系统中没有安装 MASM 或 TASM，在该软件包中还附带有 MASM 或 TASM 的常用程序。

## 4.4 MasmEdit 汇编语言集成编译器

MasmEdit 是一个由国内开发人员编写的自由软件，其中文名称为“汇编语言编译器”，可以用来编译宏汇编语言（主要是用于教学）。其主要特点是：

（1）自动定时保存文件，可设定保存的时间间隔等。

（2）同时打开多个文件，可用剪贴板在多个文件之间交换代码。

（3）自带编译器，无须安装 MASM 或 TASM。

（4）保存上次使用时所作的一切设置（包括：窗体位置及大小，自动定时保存的时间间隔等）。

MasmEdit 最好安装在符合 DOS 要求的路径下，它所处理的汇编语言源文件最好也存放在纯 DOS 目录中，否则，在使用时可能会出错。

MasmEdit 的界面如图 4.2 所示。

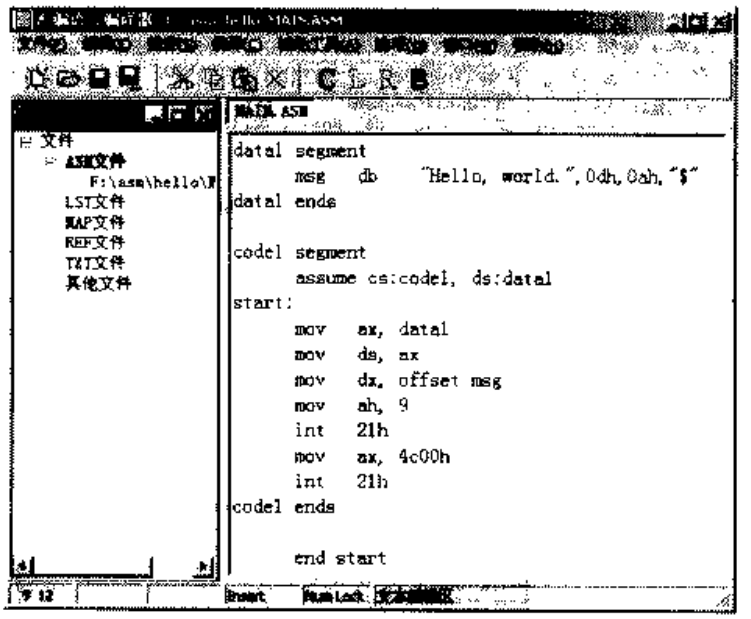



图 4.2　MasmEdit 的界面

## 4.5　EditPlus 多功能编辑器

EditPlus 是在 Internet 时代发展起来的一个可处理文本、HTML 页面及程序语言格式的 32 位编辑器。它不仅可以取代 Windows 下的文本工具，还在网页设计和程序编写方面有其独特的风格。EditPlus 可以编辑多种类型的文件，如 HTML、ASP、C/C++、Perl、Java、ASM 等文件。如果在安装时选择对应语言的语法文件，那么，EditPlus 就可以根据语法自动识别文件的关键字、标识符、字符串等内容，并用不同的颜色区别显示。

EditPlus 在文字格式的安排和设计上做得很细致，像回车、列操作等在“文档”选单中都有很详细的设定。“自动完成”功能可以使用简单缩写来实现格式字符串的完整替换功能，比如键入“if”并按空格键时，EditPlus 可将其自动扩展成“if ()”字符串，以节省程序编写时间，软件默认支持 Perl 和 C/C++语言。

EditPlus 提供了多工作窗口，可以在工作区域中打开多个文档。

在安装 EditPlus 之后，它并不能识别汇编语言语法，这时需要把 IDESetup.exe 安装到 EditPlus 所在目录。IDESetup 程序适用于编辑 C 和 ASM32 程序，其调试程序使用的是 SoftICE。对于初学者来说，SoftICE 稍微复杂了些；但对于关心软件破解、加密/解密、病毒分析的人来说 SoftICE 是一个非常重要的工具。

EditPlus 的界面如图 4.3 所示。

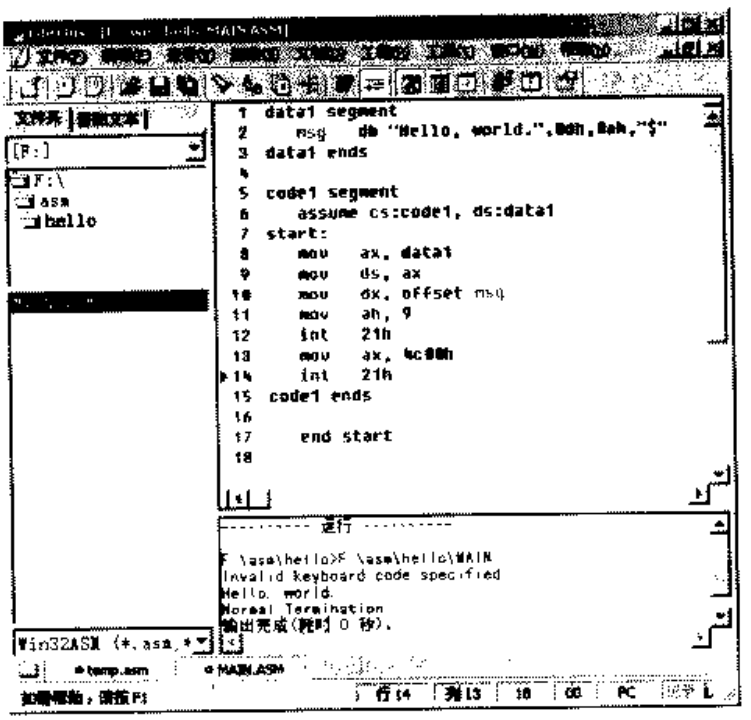



图 4.3 EditPlus 的界面

EditPlus 本身并不带编译器和调试器，它的编译和调试功能是外挂的。编译器可以使用 MASM 或 TASM，调试器可以使用 Borland C++、Turbo Debugger 或 CodeView。

以下说明如何基于 EditPlus 和 IDESetup.exe 建立 ASM16 汇编语言的集成编程环境。

假设：汇编程序使用 MASM V6.11，安装目录为 C:\MASM611，调试器使用 Borland C++ 3.1，安装目录为 c:\bc。而且 MASM 和 Borland C++ 3.1 的工作环境参数已经正确配置（配置方法见实验二和实验三）。

首先安装 EditPlus 软件包，默认情况下 EditPlus 软件包安装在 C:\Program File\EditPlus 2 目录。接下来安装 IDESetup.exe，将 IDESetup 安装在 EditPlus 软件包所在目录里，IDESetup 将覆盖 EditPlus 的部分文件。

安装完 EditPlus 和 IDESetup.exe 后，需要配置汇编、链接、跟踪、运行等方面的参数。列举如下：

（1）在“工具”->“参数选项”中设置“字体”为 Fixedsys。如图 4.4 所示。

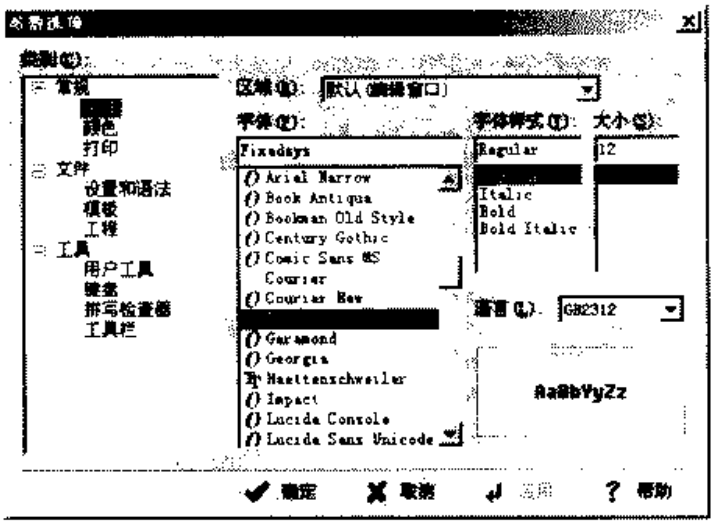



图 4.4　设置字体

（2）设置文件类型。选择“设置和语法”，IDESetup.exe 已经在文件类型中增加了一种 ASM32 或 WIN32ASM 文件类型，选中并修改之，使其“说明”设为“ASM”，“扩展名”设为“ASM”。如图 4.5 所示。

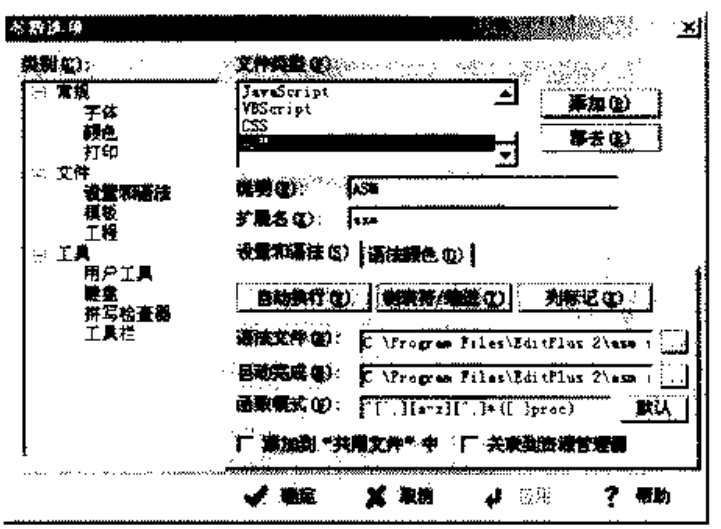



图 4.5　设置文件类型

（3）设置“汇编”工具。点击“用户工具”，将 Win32 编程工具箱中的所有工具全部删除。点击“组名”按钮，将工具组的组名改为“汇编语言编程工具箱”。点击“添加工具”

→“程序”，将菜单文本设置为“汇编”，命令设置为“C:\MASM611\BIN\MASM.EXE”，参数设置为“$(FilePath);”，设定“捕捉输出”，如图 4.6 所示。

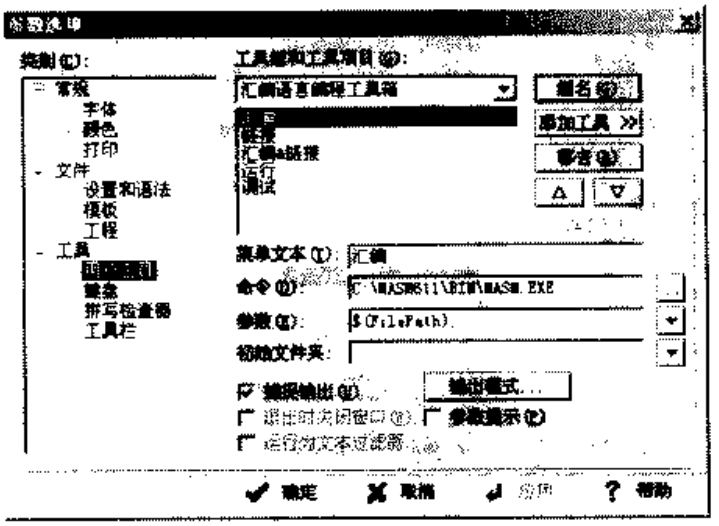



图 4.6 设置“汇编”工具

（4）设置“链接”工具。点击“添加工具”→“程序”，将菜单文本设置为“链接”，命令设置为“C:\MASM611\BINR\LINK.EXE”，参数设置为“$(FileDir)\$(FileNameNoExt).obj;”，设定“捕捉输出”，如图 4.7 所示。

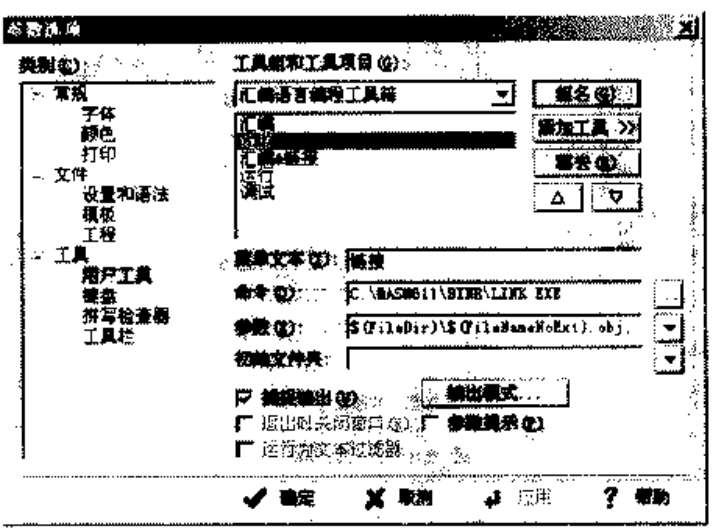



图 4.7 设置“链接”工具

（5）设置“汇编&链接”工具。点击“添加工具”→“程序”，将菜单文本设置为“汇

编&链接”，命令设置为“C:\MASM611\BIN\ML.EXE”，参数设置为“$（FilePath）”，设定“捕捉输出”，如图 4.8 所示。

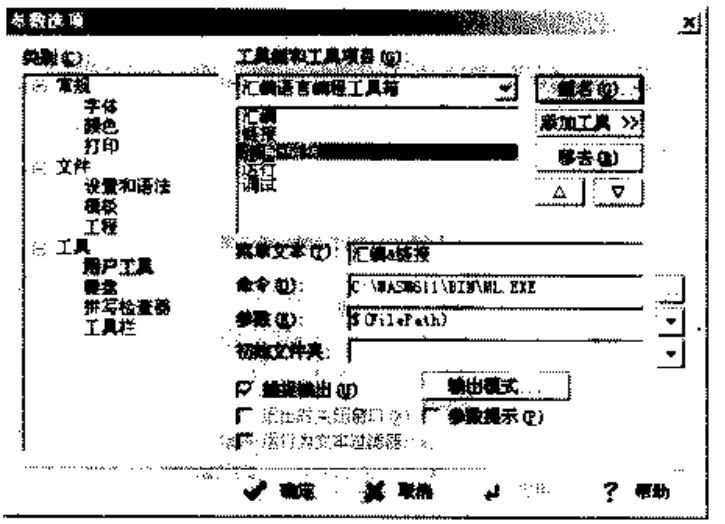



图 4.8　设置“汇编&链接”工具

（6）设置“运行”工具。先建立一个文件名为“run.bat”的批处理文件，文件内容为：%1，将其存放在 MASM 的安装目录的 BIN 子目录，即 C:\MASM611\BIN。

点击“添加工具”→“程序”，将菜单文本设置为“运行”，命令设置为“C:\MASM611\BIN\run.bat”，参数设置为“$(FileDir)\$(FileNameNoExt)”，设定“捕捉输出”，如图 4.9 所示。

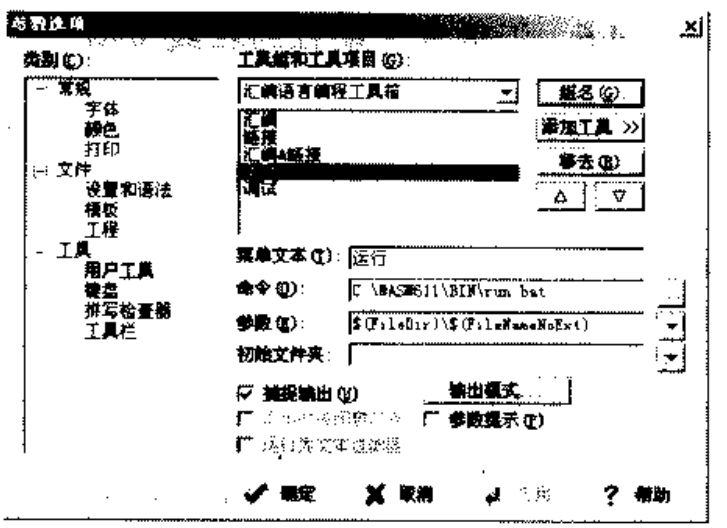



图 4.9　设置“运行”工具

（7）设置“调试”工具。点击“添加工具”→“程序”，将菜单文本设置为“调试”，命令设置为“C:\BC\BIN\TD.EXE”，参数设置为“$(FileDir)\ $(FileNameNoExt).exe”，如图4.10所示。

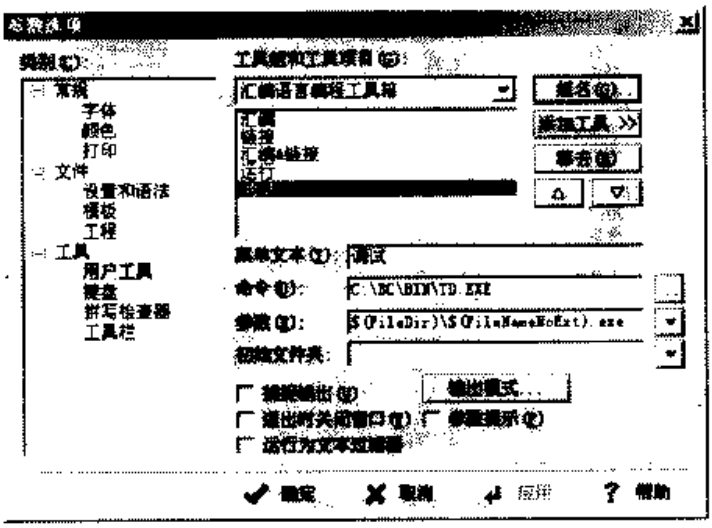



图4.10 设置“调试”工具

至此，一个基于EditPlus的中文宏汇编语言的编辑、编译、运行和调试集成环境就设置好了，它可以同时编辑和调试多个汇编语言文件，能够自动识别汇编语言语法。界面美观友好，非常适合学生和程序员编写汇编语言程序。

## 4.6 汇编语言编程的在线帮助

在汇编语言编程时，所学习和需要掌握的CPU指令和伪指令比较多，对于初学者来说要全部记下来有一定的困难。HelpPC是一个小巧简单的指令查询工具，可以提供如下在线帮助：

（1）80x86汇编语言的指令说明；

（2）C语言的指令使用；

（3）部分硬件的数据规格说明；

（4）DOS、BIOS、EMS的中断调用说明；

（5）DOS、BIOS使用的数据结构的表结构及格式。

下载HelpPC的压缩包，解压后直接运行文件“HELPPC.COM”即可。

HelpPC的界面如图4.11所示。

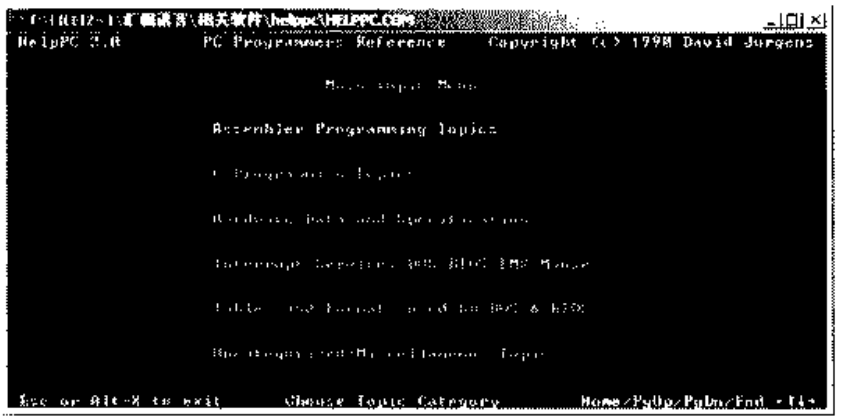



图 4.11　HelpPC 的工作界面

## 4.7　UltraEdit 多功能编辑器

UltraEdit 也是个非常有名的编辑器，它也可以像 EditPlus 那样建立一个集编辑、编译、运行和调试为一体的汇编语言集成编程环境。UltraEdit 可以说是 Windows 环境中最优秀的二进制文件编辑软件。使用汇编语言编写程序时经常需要直接跟二进制文件打交道，直接读取或存储二进制文件的文件内容，这时可以使用 UltraEdit 阅读和修改汇编语言程序的处理结果。

UltraEdit 的界面如图 4.12 所示。

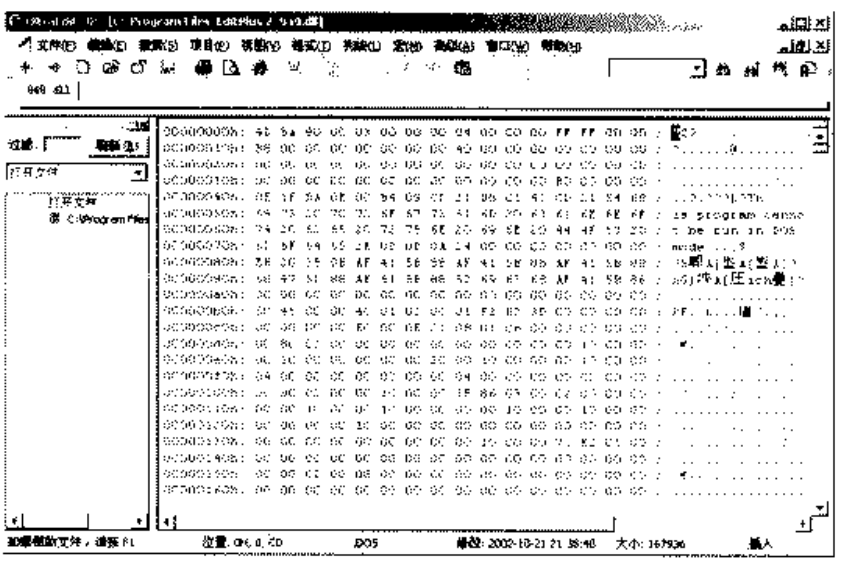


图 4.12　UltraEdit 的工作界面

## 4.8 编程环境的使用

把 EditPlus 配置成汇编语言的集成编程环境，并完成前面实验所做程序 hello.asm 的编辑、汇编、运行和调试等操作。

# 实验五　掌握操作数的各种寻址方式

## 5.1　实 验 目 的

（1）学习和掌握操作数的各种寻址方式。

（2）学习使用 DEBUG 工具调试程序。

## 5.2　预 备 知 识

1．操作数和寻址方式

操作数是指令或程序的主要处理对象。如果某条指令或某个程序不处理任何操作数，那么该指令或程序不可能有数据处理功能。在 CPU 的指令系统中，除 NOP（空操作指令）、HLT（停机指令）等少数指令之外，大量的指令在执行过程中都会涉及到操作数。所以，在指令中如何表达操作数或操作数所在位置就是正确运用汇编指令的一个重要因素。

在指令中，指定操作数或操作数存放位置的方法称为寻址方式。操作数的各种寻址方式是用汇编语言进行程序设计的基础。微机系统有 7 种基本的寻址方式：立即寻址方式、寄存器寻址方式、直接寻址方式、寄存器间接寻址方式、寄存器相对寻址方式、基址加变址寻址方式、相对基址加变址寻址方式等。其中，后五种寻址方式对应着确定内存单元有效地址的五种不同的计算方法，用它们可方便地实现对数组元素的访问。

2．传送指令 MOV（Move Instruction）

传送指令是使用最频繁的指令，它相当于高级语言中的赋值语句。指令格式如下：

MOV　Reg/Mem, Reg/Mem/Imm

Reg:Register（寄存器），Mem:Memory（存储器），Imm:Immediate（立即数），它们可以是 8 位、16 位或 32 位（特别指出其位数的除外）。在本书以后的内容里，都将采用上述缩写，不再说明。

指令的功能是把源操作数（第二操作数）的值传给目的操作数（第一操作数）。指令执行后，目的操作数的值被改变，而源操作数的值不变。当存储单元是该指令的一个操作数时，该操作数的寻址方式可以是任意一种存储单元寻址方式。

3．加法指令 ADD（ADD Binary Numbers Instruction）

指令格式：ADD　Reg/Mem, Reg/Mem/Imm。

受影响的标志位：AF、CF、OF、PF、SF 和 ZF。

指令的功能是把源操作数的值加到目的操作数中。

4. 中断 INT 20H

功能描述：终止当前正在运行的程序，它是几种终止程序运行方法之一。

入口参数：CS＝PSP 的段地址。

出口参数：无。

## 5.3 使用 DEBUG 编写汇编程序

按以下步骤输入一系列的命令和指令，编写出简单的汇编程序：

```
...>debug
-e200
xxxx:0200 xx.10 xx.20 xx.30 xx.40     (将 200H~203H 的内容改为 10H~40H)
-e150
xxxx:0150 xx.00 xx.01                 (将 150H~151H 的内容改为 00H 和 01H)
-a100
xxxx:0100 mov ax, cs                  (01)
xxxx:0102 mov ds, ax                  (02)
xxxx:0104 mov bx, 100                 (03)
xxxx:0107 mov ax, [150]               (04)
xxxx:010A add bx, ax                  (05)
xxxx:010C mov al, [bx]                (06)
xxxx:010E add al, [bx+1]              (07)
xxxx:0111 mov si, 2                   (08)
xxxx:0114 add al, [bx+si]             (09)
xxxx:0116 mov ah, 0                   (10)
xxxx:0118 mov ch, 0                   (11)
xxxx:011A mov cl, [bx+si+1]           (12)
xxxx:011D add ax, cx                  (13)
xxxx:011F int 20
```

## 5.4 跟踪汇编语言程序

（1）指出以上输入的汇编语言程序中指令（01）到指令（13）的第二操作数使用的是什么寻址方式。

（2）使用 DEBUG 的跟踪命令“P”对以上的汇编语言程序进行单步跟踪，观察指令的执行结果。

# 实验六　调试汇编源程序和 COM 可执行程序

## 6.1　实 验 目 的

（1）学习使用工具调试和修改汇编语言源程序。

（2）学习使用工具调试和修改 COM 可执行程序。

## 6.2　预 备 知 识

1．调试工具

有关 DEBUG 和 MASM 命令的使用，在实验一和实验二中有详细说明。

2．中断指令

学习汇编语言自然要接触到中断这个概念。为了循序渐进地掌握汇编语言，初学者可以暂且不去理解中断的实现方式，只需把中断看做一种实现某种功能的子程序。

所谓中断就是 CPU 暂停当前程序的执行，转而执行处理紧急事务的程序，并在该事务处理完后能自动恢复执行原先程序的过程。在此，称引起紧急事务的事件为中断源，称处理紧急事务的程序为中断服务程序或中断处理程序。计算机系统还根据所需处理事务的紧急程度，把中断分为不同的优先级，并规定：高优先级的中断能暂停低优先级的中断服务程序的执行。

3．Int 21h 的 02H 号功能

功能描述：向标准输出设备（如屏幕）输出一个字符。该输出还可被重定向，但如果这样做，将无法判断磁盘是否满。

入口参数：AH=02H，DL=待输出字符的 ASCII 码。

出口参数：无。

4．Int 21h 的 4CH 号功能

功能描述：终止程序的执行，并可返回一个代码。

入口参数：AH=4CH，AL=返回的代码。

出口参数：无。

5．无符号数除法指令 DIV

指令格式：DIV　Reg/Mem

指令的功能是用显式操作数去除隐含操作数（都为无符号数），所得商和余数按表 6.1

所示的对应关系存放。指令对标志位的影响无定义。

表 6.1 除法指令除数、被除数、商和余数的对应关系

| 除数位数 | 隐含的被除数 | 商 | 余数 | 举例 |
|---|---|---|---|---|
| 8 位 | AX | AL | AH | DIV BH |
| 16 位 | DX-AX | AX | DX | DIV BX |
| 32 位 | EDX-EAX | EAX | EDX | DIV ECX |

6．循环指令(Loop Until Complete)

循环指令 Loop 的一般格式：

Loop 标号

循环指令的功能如图 6.1 所示，其文字描述如下：

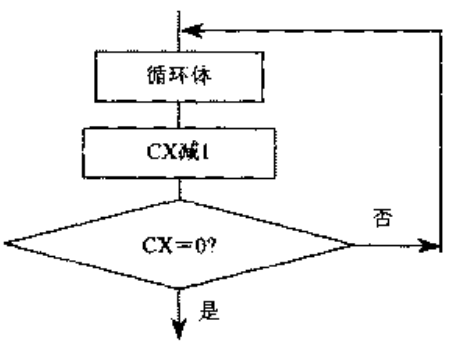



图 6.1 循环指令的功能

（1）(CX)=(CX)-1；

（2）如果(CX)≠0，转向“标号”所指向的指令；否则，终止循环，执行该指令下面的指令。

## 6.3 调试汇编语言源程序

以下汇编语言源程序用来计算 1+2+……+50 的值，并且将结果显示出来。注意：下面的汇编程序是有错误的。

```
data1 segment
    szmsg db 4 dup(?),13,10,'$'
data1 ends
code1 segment
    assume cs:code, ds:data
start:
```

```
        mov ax, data1           ;初始化段寄存器
        mov ds, ax
        mov cx, 50              ;设置循环次数
        mov ax, 0
    again:
        add ax, cx
        loop    again           ;循环累加
        mov dx, 0               ;将计算结果转化为十进制表示的字符串存放在 szmsg 中
        mov bx, 10h
        div bx
        add dl, 30h             ;计算个位的值并转化成 ASCII 码
        mov szmsg[0], dl
        mov dx, 0
        div bx
        add dl, 30h             ;计算十位的值并转化成 ASCII 码
        mov szmsg[1], dl
        mov dx, 0
        div bx
        add dl, 30h             ;计算百位的值并转化成 ASCII 码
        mov szmsg[2], dl
        add al, 30h             ;计算千位的值并转化成 ASCII 码
        mov szmsg[3], al
        mov ah, 9               ;输出计算结果
        lea dx, szmsg
        int 21h
        mov ah, 4ch             ;程序结束
        int 21h
    code1 ends
        end start
```

1. 要求

用 DEBUG、MASM 或其他工具调试该程序，找出错误并加以更正。

2. 提示

1）注意汇编语言源程序中十进制和十六进制的表示方式；

2）该程序有两个错误，需要修改 5 个地方；

3）可以使用调试工具跟踪程序的运行情况，分析程序逻辑，发现错误位置。

## 6.4　调试可执行程序

按以下步骤输入一系列的命令和指令，生成一个 COM 可执行文件，其文件名是

test1.com，它的功能是在屏幕上显示一个用 10 行“*”构成的等腰直角三角形。

```
…>debug
-a100
xxxx:0100 mov ax, cs
xxxx:0102 mov ds, ax
xxxx:0104 mov bx, 000a
xxxx:0107 mov cx, bx
xxxx:0109 mov ah, 02
xxxx:010B mov dl, 2a
xxxx:010D int 21
xxxx:010F loop 0109
xxxx:0111 mov ah, 02
xxxx:0113 mov dl, 0d
xxxx:0115 int 21
xxxx:0117 mov ah, 02
xxxx:0119 mov dl, 0a
xxxx:011B int 21
xxxx:011D dec bx
xxxx:011E jnz 0107
xxxx:0120 mov ah, 4c
xxxx:0122 int 21
-rcx
 CX 0000
:0024
-N test1.com
-w
Writing 00024 bytes
-q
…>
```

1．要求

用 DEBUG、Turbo Debugger 或其他工具修改该程序，使程序的运行结果为在屏幕上显示一个用 5 行“=”构成的等腰直角三角形，并把修改后的程序保存为 test2.com。

2．提示

1）要看懂程序首先需要掌握 DEBUG 中 A、U、G、N、W 等命令的作用；

2）字符“=”的 ASCII 码为 3dh，字符“*”的 ASCII 码为 2ah。

# 实验七　使用偏移量伪指令和联合数据类型

## 7.1　实 验 目 的

（1）学习使用偏移量伪指令。

（2）学习使用联合数据类型。

## 7.2　预 备 知 识

1．调整偏移量伪指令 ORG

调整偏移量伪指令格式：

ORG　数值表达式

伪指令的作用：使汇编程序将本伪指令下面的内存变量从该“数值表达式”所指定的偏移量开始分配。

假设有下列变量定义，且变量 word1 的偏移量为 0。

```
word1   DW  1234h
byte1   DB  56h
word2   DW  0abcdh
        ORG 1
byte2   DB  ?
word3   DW  ?
byte3   DB  ?
```

前 3 个变量定义的内存分布如图 7.1 的左边所示，但由于伪指令“ORG 1”的作用，其后面所说明的变量要从偏移量为“1”的内存单元开始存放。所以，后 3 个变量的内存分配如图 7.1 的右边所示。

由图 7.1 可知，这些变量的内存分配是相互重叠的，对某个变量的操作无疑会影响到与之重叠的变量。

另外，变量 byte2、word3 和 byte3 没有赋初值，如果赋初值的话，则重叠部分内存单元的初值将被覆盖。

2．偏移量计数器的值

汇编语言提供了一个特殊的符号“$”来引用偏移量计数器的值。

例如：

```
W1   DW   $, $
     ORG $+3  ;从当前地址开始空 3 个字节
B1   DB   43h
```

假设：在给变量 W1 分配内存单元时，当前偏移量计数器的值为 2。

于是，变量 W1 后面第 1 个“$”代表数值 2，第 1 个字分配后，此时偏移量计数器$的值就为 4，所以，第 2 个“$”就代表数值 4。

在分配完两个字之后，偏移量计数器的值变为 6，$+3 的值为 9，所以，伪指令“ORG $+3”就表示下一个变量从偏移量为 9 的单元地址开始分配。

综上分析，上述变量说明所对应的内存单元分布如图 7.2 所示。

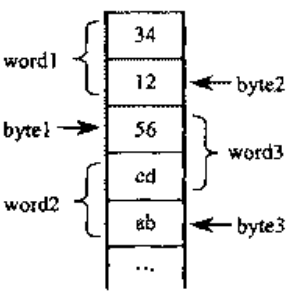



图 7.1 内存变量分配示意图

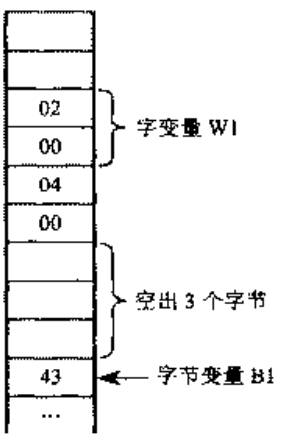



图 7.2 内存单元分布示意图

3．联合类型

联合数据类型是一种特殊的数据类型。它可以实现：以一种数据类型存储数据，以另一种数据类型来读取数据。程序员可以根据不同的需要，以不同的数据类型来读取联合类型中的数据。也就是说，在一些情况下，以一种数据类型来读取联合类型中的数据，而在另一些情况下，又以另一种数据类型来读取其数据。

联合数据类型其说明格式如下：

```
[联合类型名]   UNION   [Alignment] [,NONUNIQUE]
                数据定义语句序列
[联合类型名]   ENDS
```

联合类型中的各字段相互覆盖，即同样的存储单元被多个不同的字段所对应，并且其每个字段的偏移量都为 0。

联合类型所占的字节数是其所有字段所占字节数的最大值。

例如：

```
DATATYPE    UNION
        BB    DB   ?        ;定义一个字节
                              类型的字段
        WW    DW   ?        ;定义一个字类
                              型的字段
        DD    DD   ?        ;定义一个双字
                              类型的字段
DATATYPE    ENDS
```

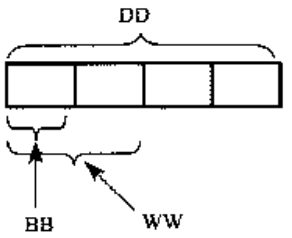



图 7.3　联合类型 DATATYPE 的字段分布示意图

联合类型 DATATYPE 的字段分布如图 7.3 所示。

## 7.3　使用偏移量伪指令编程

根据要求完成以下汇编语言程序，并调试通过：

定义一个字符串 s，其初值设置为“123456”，13，10，‘$’，该字符串是一个以‘$’结束的字符串，将其内容输出至显示器。再定义一个字节类型长度为 6 数组 a，要求 s 和 a 的偏移量相同。将 a 的内容赋值为‘a’，‘b’，‘c’，‘d’，‘e’，‘f’，然后再次将 s 的内容输出至显示器。如果正确的话，将输出“abcdef”。程序要求使用偏移量伪指令方式实现。

汇编源程序建议采用以下的基本结构：

```
data1 segment
    s db "123456",13,10,'$'
    ;(在此定义 a 的内容，使得 a 与 s 使用相同的内存偏移量)
data1 ends
code1 segment
    assume cs:code, ds:data
start:
    mov ax, data1    ;初始化段寄存器
    mov ds, ax
    mov ah,9         ;输出 s 的值
    mov dx,offset s
    int 21h
    ;(编程给数组 a 赋值)
    mov ah,9         ;再次输出 s 的值
    mov dx,offset s
    int 21h
    mov ah, 4ch      ;程序结束
    int 21h
```

```
code1 ends
    end start
```

## 7.4 使用联合类型编程

要求：

使用联合类型实现上题，依然可以使用上面的程序基本结构，只是在数据定义时不得再使用偏移量伪指令。

# 实验八　掌握汇编语言基本程序结构

## 8.1　实 验 目 的

学习汇编基本程序结构（顺序、分支、循环）编程。

## 8.2　预 备 知 识

1．闰年的判断规则

如果1个年份不能被4整除，则它不是闰年。

如果1个年份能被4整除，但不能被100整除，则它是闰年。

如果1个年份能被100整除，但不能被400整除，则它不是闰年。

如果1个年份能被400整除，则它是闰年。

2．Int 21h的09h号功能

功能描述：输出一个字符串到标准输出设备上。如果输出操作被重定向，那么，将无法判断磁盘已满。

入口参数：AH=09H，DS:DX=待输出字符串的地址，该字符串以‘$’作为其结束标志。

出口参数：无。

## 8.3　掌握分支结构编程

要求：

编写一个判断闰年的汇编语言程序，判断某个年份是否是闰年，如果是，输出“yes!”；如果不是，则输出“No!”。

年份变量名称为year，它是一个字变量。

将year依次赋值为1995、1664、1900、2000，检查程序的输出结果是否正确。

## 8.4　掌握循环结构编程

要求：

把下列C语言的语句改写成功能相同的汇编语言程序（其中的变量全部为整型），以

十进制字符串的形式输出运算结果。

```
for (i=1,j=0;i<100;i++)
    if (i%2==0) j+=i;
```

# 实验九　将数值转换成字符串

## 9.1　实 验 目 的

（1）理解二进制数和字符串的区别。

（2）掌握如何将数值转换成二进制数形式的字符串。

（3）掌握如何将数值转换成十六进制数形式的字符串。

## 9.2　预 备 知 识

Int 21h 的 02h 号功能

功能描述：向标准输出设备（如屏幕）输出一个字符。该输出还可被重定向，但如果这样做，将无法判断磁盘是否满。

入口参数：AH=02H，DL=待输出字符的 ASCII 码。

出口参数：无。

## 9.3　将数值转换成二进制形式的字符串

要求：

编写汇编语言程序将字变量类型的数值转换成二进制形式的字符串，输出结果。例如，当该数值为 256 时，程序输出“0000 0001 0000 0000”。

将该数值变量的变量名设为 value1。

依次将 value1 的值设定为 0、–1、21 845，检查转换结果是否正确。

## 9.4　将数值转换成十六进制形式的字符串

要求：

编写汇编语言程序将字变量类型的数值转换成十六进制形式的字符串，输出结果。例如，当该数值为 4 811 时，程序输出“12CB”。

将该数值变量的变量名设为 value2。

依次将 value2 的值设定为 0、–1、21 845，检查转换结果是否正确。

## 9.5 以十六进制字符串形式显示内存值（选做）

要求：

编写一段程序，以十六进制的形式显示内存 0000H:0000H 开始的 100 个字节，要求每行显示 16 个字节，每个字节之间有空格。

# 实验十　数组的使用

## 10.1　实 验 目 的

（1）学习在汇编语言中使用数组进行排序。

（2）学习如何将字符串当作数组变量进行处理。

## 10.2　预 备 知 识

冒泡排序算法是最简单的一种排序算法。它的基本思想是，将待排序的元素看做是竖着排列的“气泡”，小的元素浮在上面，而大的元素沉在下面。冒泡排序算法中需要对整个元素序列进行若干次遍历。每一次遍历，都要自底向上检查整个序列，判断两个相邻的元素的顺序是否正确。如果发现两个相邻元素的顺序不对，即小的元素在下面，就交换它们的位置。显然，处理一遍之后，最小的元素就浮到了最高位置；处理两遍之后，“次小”的元素就浮到了次高位置（在做第二遍处理时，由于最高位置上的元素已是“最小”元素，所以不必检查）。一般地，第 $i$ 遍处理时，不必检查第 $i$ 高位置以上的元素，因为经过前面 $i-1$ 遍的处理，它们已正确地排好序。

冒泡排序算法可采用如下方法实现：

```
procedure Bubble_Sort(var L:List);
var
  i,j:position;
begin
  for i:=First(L) to Last(L)-1 do
    for j:=First(L) to Last(L)-i do
      if L[j]>L[j+1] then swap(L[j],L[j+1]);   //交换L[j]和L[j+1]
end;
```

上述算法将较大的元素看做较重的气泡，每次最大的元素沉到表尾。其中 First(L)和 Last(L)分别表示线性表 L 的第一个元素和最后一个元素的位置，swap(x,y)交换变量 x,y 的值。上述算法简单地将线性表的位置当作整数用 for 循环来处理，但实际上线性表可能用链表实现；而且上述算法将线性表元素的值当作其键值进行处理。不过这些并不影响表达该算法的基本思想。

容易看出，该算法总共进行了 n(n–1)/2 次比较，因而时间复杂性为 $O(n^2)$。

## 10.3 编写冒泡排序算法程序

要求：

定义一个数组序列：DataArray dw 1567，25，46，4560，678，1234，32540。用汇编语言实现冒泡排序算法，对该数组由小到大排序，以十进制字符串的形式输出排序结果。

## 10.4 编写滚动条程序

要求：

编写汇编语言程序，将一个字符串以单词为单元循环显示。原字符串是“I am glad to see you.”，字符循环效果如下：

“I am glad to see you.”

“am glad to see you. I”

“glad to see you. I am”

“to see you. I am glad”

“see you. I am glad to”

“you. I am glad to see”

“I am glad to see you.”

……

提示：

（1）将字符串当作字符数组处理；

（2）将空格和“.”作为单词开始和结束的依据；

（3）建议使用一个与原字符串相同大小的字符数组做缓冲区，再使用两个偏移量指针标识循环的开始位置和下一次循环的开始位置。

# 实验十一　编写子程序

## 11.1　实 验 目 的

（1）学习子程序的编写方法。

（2）学习子程序的参数传递方法。

（3）理解递归算法的作用。

## 11.2　预 备 知 识

递归算法

为了描述问题的某一状态，必须用到它的上一状态，而描述上一状态，又必须用到它的上一状态。这种用自己来定义自己的方法，称为递归定义。

例如：阶乘 n!=n*(n–1) *(n–2) *...*2*1，可以定义函数 f(n)为：

$$\begin{cases} f(n)=f(n-1)*n;\ (n>1) \\ f(1)=1; \end{cases}$$

由上例可看出，递归定义有两个要素：

（1）递归边界条件。也就是所描述问题的最简单情况，它本身不再使用递归的定义。如上例中当 n=1 时，f(n)=1，不使用 f(n–1)来定义。

（2）递归定义：使问题向边界条件转化的规则。递归定义必须能使问题越来越简单。如上例中 f(n)由 f(n–1)定义，越来越靠近 f(0)，即边界条件。最简单的情况是 f(0)=1。

递归算法的效率往往很低，既费时又费内存空间。但是递归也有其长处，它能使一个蕴含递归关系且结构复杂的程序简单精炼，增加可读性，特别是在难于找到从边界到解的全过程的情况下，如果把问题推进一步来得出结果且仍维持原问题的关系，则采用递归算法编程比较合适。

递归按其调用方式分为：

（1）直接递归：递归过程 P 直接自己调用自己。

（2）间接递归：P 包含另一过程 D，而 D 又调用 P。

递归算法适用的一般场合：

（1）数据的定义形式按递归定义，如阶乘。

（2）数据之间的关系（即数据结构）按递归定义，如树的遍历，图的搜索等。

（3）问题解法按递归算法实现，如回溯法等。

## 11.3 使用寄存器向子程序传递参数

要求：

编写一个计算阶乘的子程序，该子程序可以计算 8 以内数值的阶乘。子程序通过寄存器 BL 接受参数，通过寄存器 AX 输出计算结果。

除了 BX 和 AX 两个寄存器之外，子程序调用前后其他通用寄存器的值不得改变。

主程序调用阶乘子程序依次计算 1~7 的阶乘值并以十进制的形式输出结果。

## 11.4 使用约定存储单元向子程序传递参数

要求：

编写一个将字节变量转换成规定格式的字符串的子程序。字节变量被转换成两个字节的十六进制形式的字符串，字符串以一个空格字符结尾。假设字节变量为 0x3A 时，输出“3A”。

子程序通过寄存器 BL 接受参数，通过指定的存储单元 output db 3 dup(?)输出计算结果。

定义字节变量 DDD DB 30H，41H，52H，63H，74H，25H，36H，47H，依次调用子程序将各个值转换成字符串后连续输出。

除了 BX 和 AX 两个寄存器之外，子程序调用前后其他通用寄存器的值不得改变。

## 11.5 使用堆栈传递参数的方法编写递归子程序(*)

裴波那契数列的定义：

（1）f(0)=1，f(1)=2；

（2）f(n)=f(n-1)+f(n-2)。

计算裴波那契数列的递归程序：

```
function fib(n: integer): integer;
begin
    if n = 0 then fib := 1
    else if n = 1 then fib := 2
         else fib := fib(n-2) + fib(n-1);
End;
```

要求：

编写汇编语言子程序实现该递归算法，调用该子程序计算 fib(10)，并将结果以十进制字符串的形式显示出来。

输入和输出参数的传递只能使用堆栈。

要保护各种通用寄存器的值。

提示：

使用堆栈传递参数可以参考下例：

```
sum proc                          ;sum 子程序用于计算 3 个字变量的和，结果存于 ax。
    push    bp
    mov     bp, sp
    mov     ax, word ptr [bp+4]
    add     ax, word ptr [bp+6]
    add     ax, word ptr [bp+8]
    pop     bp
    ret
sum endp
```

# 实验十二　键盘输入输出处理

## 12.1　实 验 目 的

学习使用 DOS 功能调用、BIOS 中断编写键盘处理程序。

## 12.2　预 备 知 识

1．中断的分类

汇编语言程序员能够控制的软件中断的功能及其使用方法，常用的这类中断有 DOS 功能调用(INT 21H)、BIOS 中断、硬件和外设的中断等。

在用户程序中，若直接通过端口来操作硬件或外设，那么，其处理过程没有额外的多余工作，处理速度显然是最快的，但这样做，无疑使得用户程序具有很大的局限性。硬件环境的改变将直接影响程序的正常运行。

若用户程序通过调用 DOS 功能来实现其所需功能，那么，应用程序与低层硬件则会相距较远，操作最终的对象需要经过中间环节，处理速度肯定会受到一定的损失，但这种应用程序适应性强，应用范围广，对硬件的依赖性较小。

由于 BIOS 介于 DOS 和具体硬件之间，所以，调用 BIOS 的功能是一个很好的折中方案。程序员可在以下 3 种情况下考虑使用 BIOS 的功能：

（1）BIOS 提供的功能，而 DOS 没有提供该功能的情况；

（2）不能利用 DOS 功能调用的情况(可能因为某些具体应用的限制)；

（3）基于处理速度的考虑，需要绕过 DOS 层的情况。

综上所述，可以归纳出如下结论：使用中断的层次越高，它与硬件设备相关程度就越低，处理速度也就越低，但用户程序的适用范围较广，反之亦然。

程序员可根据应用程序的要求、对硬件环境的熟悉程度等因素来选用不同层次的中断。

2．键盘状态字

在计算机键盘上除了可输入各种字符（字母、数字和符号等）的按键之外，还有一些功能键（如 F1、F2 等）、控制键（如 Ctrl、Alt、Shift 等）、双态键（如 Num Lock、Caps Lock 等）和特殊请求键（如 Print Screen、Scroll Lock 等）。

键盘中的控制键和双态键是非打印按键，它们是起控制或转换作用的。当使用者按下

控制键或双态键时，系统要记住其所按下的按键。为此，在计算机系统中，特意安排一个字来标志这些按键的状态，称该字为键盘状态字。

键盘状态字的各位含义如图 12.1 所示。

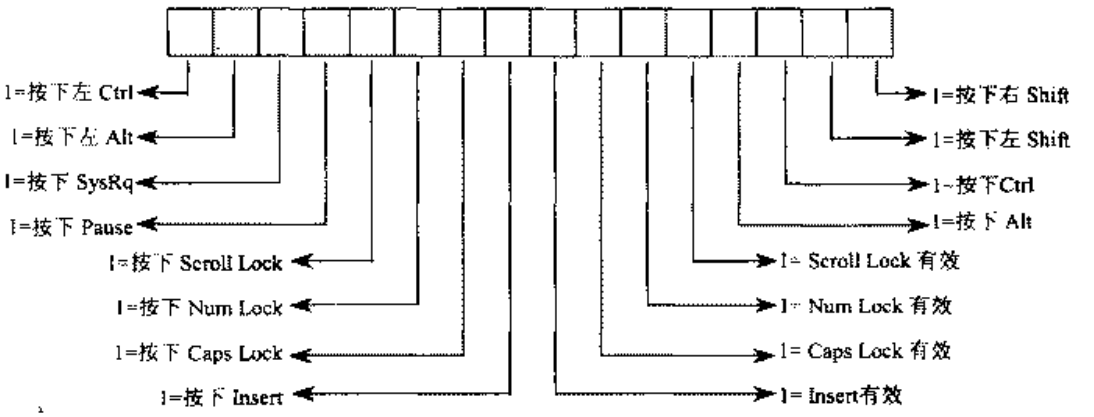



图 12.1　键盘状态字的各位含义

3．键盘中断的处理过程

当用户按键时，键盘接口会得到一个代表该按键的键盘扫描码，同时产生一个中断请求。键盘中断服务程序先从键盘接口取得按键的扫描码，然后根据其扫描码判断用户所按的键并做相应的处理，最后通知中断控制器本次中断结束并实现中断返回。

（1）若用户按下双态键（如 Caps Lock、Num Lock 和 Scroll Lock 等），则在键盘上相应 LED 指示灯的状态将发生改变；

（2）若用户按下控制键（如 Ctrl、Alt 和 Shift 等），则在键盘标志字中设置其标志位；

（3）若用户按下功能键（如 F1、F2 等），再根据当前是否又按下控制键来确定其系统扫描码，并把其系统扫描码和一个值为 0 的字节存入键盘缓冲区；

（4）若用户按下字符键（如 A、1、+等），此时，再根据当前是否又按下控制键来确定其系统扫描码，并得到该按键所对应的 ASCII 码，然后把其系统扫描码和 ASCII 码一起存入键盘缓冲区；

（5）若用户按下功能请求键（如 Print Screen 等），则系统直接产生一个具体的动作。

4．DOS 键盘中断

INT 21H 中的相关功能如下：

（1）01H——带回显的键盘输入；

（2）06H——控制台的输入/输出：当 DL=0FFH，表示键盘输入；

（3）07H——不回显、不过滤的键盘输入；

（4）08H——不回显的键盘输入；

（5）0AH——键盘输入字符串；

（6）0BH——检查键盘输入状态；

（7）0CH——清除输入缓冲区的输入功能。

5．BIOS 键盘中断

在 BIOS 系统中，提供了中断 16H 来实现键盘输入功能。其具体的功能如下：

（1）00H、10H——从键盘读一个字符，输入字符不回显；

（2）01H、11H——判断键盘缓冲区内是否有字符可读；

（3）02H——读取当前键盘状态字。

## 12.3 使用 DOS 中断接收和处理键盘输入

要求：

编写汇编语言程序，使用 DOS 中断接收键盘输入。每次以不回显的方式接收一个字符，将其以十六进制字符串的形式显示。例如，键盘输入“1”时，屏幕输出“30”。

当接收到 Esc 键时，程序结束。

## 12.4 使用 BIOS 中断接收和处理键盘输入

要求：

编写汇编语言程序，使用 BIOS 中断接收和显示键盘输入。只要求能够处理控制键（如 Ctrl、Alt 和 Shift 等）和大小写字符键（如 A、z 等）。显示方式如下：

（1）当按下 A、b、z 等字符键时，就显示字符本身；

（2）当同时按下控制键和字符键时，显示控制键标记及字符。例如：

① 按下 Ctrl 键和 B 时，显示 Ctrl-B；

② 按下 Alt 键和 k 时，显示 Alt-k；

③ 按下 Ctrl、Shift 键和 z 时，显示 Ctrl-Shift-z；

④ 按下 Ctrl、Alt 键和 z 时，显示 Ctrl-Alt-z；

⑤ 按下 Shift、Alt 键和 z 时，显示 Alt-Shift-z；

⑥ 按下 Ctrl、Shift、Alt 键和 z 时，显示 Ctrl-Alt-Shift-z；

（3）对于 Ctrl 键、Shift 键和 Alt 键，不要求区分左、右键；

（4）当按下 Esc 键时，程序结束；

（5）不要求对双态键（如 Caps Lock、Num Lock 和 Scroll Lock 等）、功能键（如 F1、F2 等）和功能请求键（如 Print Screen 等）处理。

# 实验十三　图形模式编程

## 13.1　实 验 目 的

学习使用汇编语言进行图形模式编程。

## 13.2　预 备 知 识

1．图形模式

计算机系统中的显示器都有两种显示方式：文本显示方式和图形显示方式。

图形显示的最小单位是像素，对每个像素可用不同的颜色来显示。所以，在显示缓冲区内记录的信息是屏幕各像素的显示颜色。

由于各种图形显示模式所能显示的颜色和像素是不同的，它决定了显示缓冲区的存储方式也是不同的。

2．BIOS 中的屏幕操作中断调用

BIOS 系统提供了中断 10H 来实现各种屏幕处理功能。其具体的功能如下：

（1）00H——设置显示器模式；

（2）01H——设置光标形状；

（3）02H——设置光标位置；

（4）03H——读取光标信息；

（5）05H——设置显示页；

（6）06H、07H——初始化或滚屏（向上和向下滚屏）；

（7）08H——读光标处的字符及其属性；

（8）09H——在当前光标处按指定属性显示字符；

（9）0AH——在当前光标处显示字符；

（10）0CH——写图形像素；

（11）0DH——读图形像素；

（12）0EH——在 Teletype 模式下显示字符；

（13）0FH——读取显示器模式；

（14）10H——颜色；

（15）13H——在 Teletype 模式下显示字符串。

## 13.3 简单图形绘制

要求一：

将屏幕设置成 640×480×16 色图形模式，底色为黑色。在屏幕（50，50）到（200，200）处画一个实心的红色正方形。输入任意键时程序结束。

要求二：

将屏幕设置成 320×200×256 色图形模式，采用中断调用的方式，从屏幕的最上方到最下方画 200 道横线，线的颜色从 1 开始依次加 1。输入任意键时程序结束。

## 13.4 使用鼠标和键盘 BIOS 中断编程(*)

要求：

编写一个控制光标位置和形状的程序，该程序具有以下功能：

（1）可用光标移动键↑、↓、←和→来移动光标；

（2）当光标已在第 0 列，且按←键时，光标定在上一行的最后一列；若已在屏幕的左上角，则光标不动；按→键时的边界处理类似；

（3）当光标在第 0 行，且按↑键时，则光标不动；按↓键时的边界处理类似；

（4）按 Home 键或 End 键，则光标移到当前行的行首或行尾；

（5）若按下数字键或字母键，则把该字符从当前位置依次显示到屏幕顶（在新位置显示当前字符时，原位置的符号被抹去）；

（6）按 Esc 键，程序结束。

# 实验十四　鼠标中断处理

## 14.1　实 验 目 的

学习在汇编语言中编写程序鼠标处理子程序。

## 14.2　预 备 知 识

1．鼠标中断的常用功能

BIOS 系统提供了中断 33H 来实现鼠标中断功能，其常用的功能如下：

（1）00H——初始化鼠标；

（2）01H——显示鼠标指针；

（3）02H——隐藏鼠标指针；

（4）03H——读取鼠标位置及其按钮状态；

（5）04H——设置鼠标指针位置；

（6）05H——读取鼠标按键信息；

（7）06H——读取鼠标按钮释放信息；

（8）07H——设置鼠标水平边界；

（9）08H——设置鼠标垂直边界；

（10）09H——设置图形鼠标形状；

（11）0AH——设置本文鼠标形状；

（12）0BH——读取鼠标移动计数；

（13）0CH——为鼠标事件设置处理程序；

（14）0FH——设置鼠标计数与像素比；

（15）10H——设置鼠标指针隐藏区域；

（16）14H——替换鼠标事件中断；

（17）18H——为鼠标事件设置替换处理程序；

（18）19H——读取替换处理程序的地址；

（19）1DH——为鼠标指针选择显示页；

（20）1EH——读取鼠标指针的显示页。

2．鼠标指针的设置

鼠标指针主要有两种表现形式：文本鼠标和图形鼠标。文本鼠标又分为软指针和硬指针。软指针是用各种字符来作为鼠标指针，而硬指针是用方块光标的大小来表示鼠标指针，其效果类似于 INT 10H 之功能 01H——设置光标类型。

## 14.3 文本鼠标编程

要求：

在文本模式下显示鼠标，在屏幕的左下角显示鼠标的坐标。当按下鼠标右键时停止程序运行。

提示：

在屏幕上绘图时需要隐藏鼠标，绘图结束后再显示鼠标。

## 14.4 图形鼠标编程

要求：

在图形模式下显示鼠标，以鼠标坐标位置为基准点在屏幕上画一条纵贯屏幕的垂直直线和一条横贯屏幕的水平直线。当鼠标移动时，两条直线也跟随移动。当按下鼠标右键时停止程序运行。

提示：

使用 12H 图形模式。

在屏幕上绘图时需要隐藏鼠标，绘图结束后再显示鼠标。

# 实验十五　宏的定义和使用

## 15.1　实 验 目 的

（1）了解宏和子程序之间的区别。

（2）学习定义和使用宏。

## 15.2　编写字符串处理的宏

要求：

编写一个字符串处理的宏，它的作用是将字符串反转。例如，把字符串“ABCD”反转成“DCBA”。该宏的参数就是字符串的地址。字符串以 0 作为结尾。

调用编写好的宏，用字符串“How do you do?”来验证。

## 15.3　编写数值计算的宏

要求：

编写一个数值计算的宏，它的作用是计算从 1+2+…+n 的值。

编写主程序调用该宏，计算当 n=50 和 n=100 时的计算结果，并以十进制字符串的形式输出。

# 实验十六　磁盘和文件操作

## 16.1　实 验 目 的

学习使用 DOS 中断对磁盘和文件进行操作。

## 16.2　预 备 知 识

1．中断 21h 的文件操作功能类

（1）文件操作功能(File Operation Function)(使用文件控制块 FCB 实现)

0FH——打开文件；　　10H——关闭文件；

11H——查找第一个文件；　　12H——查找下一个文件；

13H——删除文件；　　16H——创建文件；

17H——文件换名；　　23H——读取文件的大小；

29H——分析文件名。

（2）记录操作功能(Record Function)

1AH——设置数据传输区地址；　　2FH——读取数据传输区地址；

3FH——读文件或设备；　　40H——写文件或设备；

42H——设置文件指针；　　5CH——文件区域加锁或解锁；

68H——提交文件缓冲区数据。

（3）记录操作功能(Record Function) (使用文件控制块 FCB 实现)

14H——顺序读；　　15H——顺序写；

21H——随机读；　　22H——随机写；

24H——设置相对记录数；　　27H——随机读块；

28H——随机写块。

2．中断 21H 的磁盘操作功能类

磁盘管理功能(Disk-Management Function)

0DH——磁盘复位；　　0EH——选择磁盘；

19H——读取当前驱动器；　　1BH，1CH——读取驱动器数据；

2EH——设置校验标志；　　36H——读取驱动器分配信息；

54H——读取校验标志。

3．文件控制块 FCB(File Control Block)

在 MS DOS 操作系统中，文件控制块的字段如表 16.1 所示。

表 16.1　文件控制块的字段偏移量及字段含义

| 字　段　名 | 偏　移　量 | 字 段 含 义 |
|---|---|---|
| 驱动器字段 | 00H | 0=默认，1 for drive A, 2 for drive B，…… |
| 文件基本名 | 01H | |
| 文件扩展名 | 09H | |
| 当前块字段 | 0CH | |
| 记录大小字段 | 0EH | |
| 文件长度字段 | 10H | 文件字节数 |
| 日期字段 | 14H | 上次访问的日期 |
| 时间字段 | 16H | 上次访问的时间 |
| 保留 | 18H | |
| 当前块中记录 | 20H | |
| 随机访问的记录号 | 21H | |

## 16.3　编写文件复制程序

要求：

在 C 盘根目录下打开文件 autoexec.bat，并将其内容写到 autoexec.out 文件中。

提示：

（1）首先要尝试打开文件 autoexec.bat，如果文件不存在要报告错误。

（2）文件 autoexec.bat 打开后要创建 autoexec.out。

（3）反复对 autoexec.bat 顺序读，然后对 autoexec.out 顺序写，直到文件写完。

（4）最后关闭两个文件。

（5）在复制文件前需要调用 int 21h 的 1AH 子功能设置磁盘传送区地址。

（6）为简化起见，FCB 可以按照以下方式定义：

```
fcb   db 0,'filename', 'ext',26dup(0)
```

（7）采用 FCB 方式读写文件必须设置数据传输区地址，通过数据传输区进行文件读写。

## 16.4 编写程序检查磁盘状态

要求：

用中断 int 21h 的 36H 子功能取磁盘未用空间和总空间，并计算出已用空间，将这些信息显示在屏幕上。

提示：

（1）为磁盘空间的计算公式（以字节为单位）：

总空间=CX×AX×DX

自由空间=CX×AX×BX

使用空间= CX×AX×(DX-BX)

（2）将计算结果以十进制字符串的形式显示出来。

# 实验十七　修改中断(*)

## 17.1　实验目的

（1）学习使用 DOS 中断对文件进行操作。

（2）学习编写中断程序。

## 17.2　预备知识

1．Windows 98/NT/2000/XP 的中断向量表的读写问题

中断向量表是一个特殊的线性表，它保存着系统中所有中断服务程序或系统参数的入口地址（偏移量和段地址）。在微机系统中，该表有 256 个中断向量(0~0FFH)，每个中断向量占 4 个字节，前两个字节是中断向量的偏移量，后两个字节是中断向量的段地址。256 个中断向量总共占 1K 个字节。中断向量表的内存起始在 0000H:0000H，程序可以通过中断 21H 的 35H 功能读取中断向量，也可以通过直接访问对应的内存空间获取中断向量。

若直接访问中断向量表，假定中断号为 n，则该中断在中断向量表中的存储位置是 4n。

Windows 98/NT/2000/XP 操作系统允许通过中断调用的方式和直接访问的方式读取中断向量，但不支持修改中断向量。无论程序通过中断方式还是直接访问方式，系统将拒绝对中断向量表的任何写操作，并提示发生 NTVDM 异常。NTVDM 是 32 位 Windows 操作系统为 16 位程序建立的虚拟 DOS 机，在多线程的 32 位 Windows 操作系统中，NTVDM 以单线程模式运行。NTVDM 进程模仿 16 位的 Windows 环境。基于 MS DOS 的程序在自己的 NTVDM 进程中运行。发生 NTVDM 现异常后，程序将被强行中止。即便如此，NTVDM 的工作环境还是会发生混乱，如果运行 MEM 或其他外部命令，会发现运行结果不正确甚至死机。

因此，调试中断修改程序不适于使用 32 位 Wndows 操作系统环境，最好在纯 DOS 环境中进行。

2．中断 21h 的 25H 功能

功能描述：设置中断向量表。

入口参数：AH=25H，AL=中断号，DS:DX=中断处理程序的入口地址。

出口参数：无。

3．中断 21h 的 35H 功能

功能描述：取指定中断号的入口地址。

入口参数：AH=35H，AL=中断号。

出口参数：ES:BX=中断处理程序的入口地址。

## 17.3 编写中断处理程序

要求：

（1）编写一个中断处理程序，输入参数为 DL。该程序将 DL 的值转换成十六进制表示的字符串（后跟一个空格符）显示在屏幕上。例如，当 DL=37H 时，显示结果为“37”。

（2）保留中断 21h 的 02h 号功能的入口地址，用刚才的中断处理程序替换中断 21h 的 02h 号功能。（注意：替换的仅仅是 02h 号功能，其他的功能不得影响。因此，新编中断处理程序必须判断 ah 的值是否为 02h，如果不是，需要用一个远程跳转调用回原先的中断处理程序。）

（3）调用被替换的中断 21h 的 02h 号功能显示字符串“10Az”，检查显示效果。

（4）程序退出前恢复原有的中断 21h。

# 实验十八　驻留程序 TSR(*)

## 18.1　实 验 目 的

学习使用汇编语言中编写内存驻留程序 TSR(Terminal but Stay Resident)。

## 18.2　预 备 知 识

1．Windows 98/NT/2000/XP 的中断向量表的读写问题

调试内存驻留程序不适于使用 32 位 Wndows 操作系统环境，最好在纯 DOS 环境中进行。详细原因见实验十七的预备知识。

2．内存驻留程序 TSR(Terminal but Stay Resident)

驻留程序 TSR 是一种特殊应用程序，它在装入内存运行结束后，其部分代码仍然驻留在内存，当该段代码被激活时，它又进入运行状态。常用的驻留程序是作为某个中断处理程序的一部分，其激活条件就是系统产生了此中断的中断请求。

虽然驻留程序可根据具体的需要有不同的编写方式，但其典型结构包括以下几部分：

（1）保存、修改中断向量表；

（2）程序第一次运行时的初始化部分：

1）用自己定义的地址来取代中断向量表中的原地址；

2）确定驻留代码部分的字节数；

3）用中断 21H 之功能 31H 把需要驻留代码部分驻留在内存。

（3）驻留内存的代码部分。

3．中断 21h 的 31H 功能

功能描述：终止程序的运行，传递一个返回代码给其父进程，但该程序部分或全部驻留在内存中。

入口参数：AH=31H，AL=返回代码号，DX=驻留在内存中的字节数。

出口参数：无。

4．中断 21h 的 1AH 功能

功能描述：设置磁盘传送数据区地址，为随后 FCB 相关操作作准备。

入口参数：AH=1AH，DS:DX=指定文件控制块的地址。

出口参数：无。

5．中断 21h 的 3BH 功能

功能描述：用指定的驱动器和路径设置为当前目录。

入口参数：AH=3BH，DS:DX=指定路径的字符串地址（以 0 为字符串的结束标志）。

出口参数：CF=0——设置成功，否则，AX=错误号(03H)。

6．中断 21h 的 16H 功能

功能描述：在当前目录中创建一个文件，其文件长度为 0，并打开该文件，为随后的读/写操作做好必要的准备。

入口参数：AH=16H，DS:DX=未打开的文件控制块的地址。

出口参数：AL=00H——创建成功，否则，AL=FFH(如磁盘满)。

7．中断 21h 的 15H 功能

功能描述：向文件写入下一个顺序数据块，并相应增加文件的指针。

入口参数：AH=15H，DS:DX=先前打开文件控制块的地址。

出口参数：AL=00H——写入成功；AH=01H——磁盘满；AH=02H——段缠绕。

8．中断 21h 的 10H 功能

功能描述：关闭文件。

入口参数：AH=10H，DS:DX=文件控制块的地址。

出口参数：AL=00H——关闭成功，否则，AL=FFH。

## 18.3 编写 TSR 程序

要求：

编写一个内存驻留程序，该程序截取中断 21h 的 01 号功能，接收所有的按键信息，以追加的形式写入文件 c:\keyboard.txt。要求程序不得影响原有中断 21h 的 01 号功能。

# 实验十九　综合练习一

## 19.1　实 验 目 的

综合使用文件操作和数据转换的知识。

## 19.2　保存中断向量表

要求：

读取中断向量表，将内容存放在 C:\vector.txt 中，存储格式如下：

INT 00H　　00A7:1068

INT 11H　　0070:108B

…

INT FFH　　0202:F000

说明：

（1）每个中断用一行文字描述；

（2）“INT xxH”以十六进制字符串形式指明中断号；

（3）“xxxx:yyyy”指明中断的入口地址，xxxx 是其段地址，yyyy 是其偏移量。

提示：

（1）文件读写的有关资料可参考实验十六。

（2）可以使用直接访问的方式读取中断向量表。

（3）中断向量表的起始位置在 0000H:0000H，每个中断占 4 个字节，总共 256 个中断占用 1K 个字节的存储空间。

## 19.3　视觉效果处理

要求：

编写一个程序，它把屏幕上的数字改为蓝绿色背景的红字。

提示：

（1）程序工作在文本模式；

（2）不需要修改屏幕显示模式；

（3）可使用中断 10H 读写显示缓冲区的字符。

# 实验二十　综合练习二

## 20.1　实 验 目 的

综合使用汇编语言编写程序。

## 20.2　编写简单加密程序

要求：

编写一个简单的加密程序，它把一个文本文件中的所有字母字符按下列规律进行转换，非字母的字符保持不变。

转换规律：‘A’→‘Z’，‘B’→‘Y’，…

说明：

（1）文本文件可以自行创建，文件名可以自定，例如 a.txt；

（2）除了‘A’～‘Z’及‘a’～‘z’的字母外，其他的字符不要改动。

提示：

（1）文件读写的有关资料可参考实验十六。

（2）可以建立一个临时文件存储被转换文本内容，转换完成后删除原先的文本文件，再将临时文件更名。

## 20.3　编写光标控制程序

要求：

编写一个控制光标位置的程序，该程序具有以下功能：

（1）可用光标移动键↑、↓、←和→来移动光标；

（2）当光标已在第 0 列，且按←键时，光标定在上一行的最后一列；若已在屏幕的左上角，则光标不动，且给出响铃；按→键时的边界处理类似；

（3）当光标在第 0 行，且按↑键时，则光标不动，且给出响铃；按↓键时的边界处理类似；

（4）按 Home 键或 End 键，则光标移到当前行的行首或行尾；

（5）若按下数字键或字母键，则把该字符从当前位置依次显示到屏幕顶（在新位置显示当前字符时，原位置的符号被抹去）；

（6）按 Esc 键，程序结束。

# 第二篇 习 题 集

## 第一章 预 备 知 识

1．汇编语言的主要特点有哪些？

**答：**

汇编语言相对于机器语言有以下几个主要特点：

（1）可读性高。汇编语言采用通俗易懂的、具有一定含义的符号指令，可读性远高于二进制编码的机器语言；

（2）容易理解和维护。

汇编语言相对于高级语言有以下几个主要特点：

（1）机器相关性。汇编语言指令是机器指令的一种符号表示，而不同类型的CPU有不同的机器指令系统，也就有不同的汇编语言；

（2）执行高效率。因为汇编语言有“与机器相关性”的特性，程序员用汇编语言编写程序时，可充分发挥自己的聪明才智，对机器内部的各种资源进行合理的安排，让它们始终处于最佳的使用状态，从而达到执行代码短，执行速度快；

（3）编程较复杂。由于汇编语言程序要安排运算的每一个细节，这就使得编写汇编语言程序比较繁琐、复杂；

（4）调试较复杂。调试汇编语言程序要清楚机器资源的具体细节，大量使用转移语句，使得程序的调试比较复杂。此外，实现同样功能的汇编语言程序比高级语言程序通常要长得多，这也增加了调试的复杂程度。

2．汇编语言适用于哪些领域？在哪些领域使用不合适？

**答：**

汇编语言适用的领域包括：

（1）要求执行效率高、反应快的领域，如操作系统内核、工业控制、实时系统等；

（2）系统性能的瓶颈或频繁被使用的子程序或程序段；

（3）与硬件资源密切相关的软件开发，如设备驱动程序等；

（4）受存储容量限制的应用领域，如家用电器的计算机控制功能等；

（5）没有适当的高级语言开发环境。

汇编语言不宜使用的领域包括：

（1）大型软件的整体开发；

（2）没有特殊要求的一般应用系统的开发等。

3．在汇编语言中，如何表示二进制、八进制、十进制和十六进制的数值？

**答：**

汇编语言中，在书写数值时，按下列规则表示不同进制：

（1）书写二进制数时，在数据后面紧跟一个字母B；

（2）书写八进制数时，在数据后面紧跟一个字母Q；

（3）书写十进制数时，在数据后面紧跟一个字母D；

（4）书写十六进制数时，在数据后面紧跟一个字母H。

4．在计算机中，如何表示正、负数？在保持数值大小不变的情况下，如何把位数少的二进制数值扩展成位数较多的二进制数值？

**答：**

在计算机内，为了表示正、负数，对有符号数采用二进制的补码表示形式。补码的最高位用来表示正、负数：0表示正数，1表示负数。正数的补码是其自身的二进制形式，负数的补码是把其正数的二进制编码变“反”，再加1而得。

把位数少的二进制数值扩展成位数较多的二进制数值的具体方法就是把位数少的二进制数值的最高位扩展到位数较多的二进制数值的所有更高位。如题图1.1所示。




题图 1.1 8 位扩展为 16 位的示意图

5．在 ASCII 表中，字符“0”～“9”与数值 0~9 之间编码规律是什么？大写字母和小写字母之间的编码规律是什么？

**答：**

在ASCII表中，字符“0”～“9”的编码是连续的，且“0”的ASCII编码为30H。

英文字母在ASCII码中的编码也是连续的，大写字母的编码小于小写字母的编码，且大小写字母的编码之间相差20H。

6．汇编语言中的基本数据类型有哪些？它与高级程序设计语言（如 C 语言）中的数据类型的对应关系是什么？

**答：**

汇编语言的基本数据类型有字节、字、双字、四字和十字节等。它与高级程序设计语言（如C语言）中的对应数据类型分别为char、int、long和浮点数类型(float,double)等。

# 第二章　CPU 资源和存储器

1. 在 16 位 CPU 中，有哪些 8 位寄存器和 16 位寄存器？哪些 16 位寄存器可分为两个 8 位寄存器来使用？

**答：**

在 16 位 CPU 系统中，汇编语言程序可访问的寄存器有：

8 位寄存器：AH、AL、BH、BL、CH、CL、DH 和 DL；

16 位寄存器：4 个数据寄存器：AX、BX、CX 和 DX；

2 个变址寄存器：DI 和 SI；

2 个指针寄存器：SP 和 BP；

4 个段寄存器：ES、CS、SS 和 DS；

1 个指令指针寄存器：IP；

1 个标志寄存器：FLAGS。

可分为 2 个 8 位寄存器的是：AX、BX、CX 和 DX。

2. 同 16 位 CPU 相比，在 32 位 CPU 中，扩展了哪些寄存器？哪些寄存器的位数没变？所增加的寄存器是什么？

**答：**

在 32 位 CPU 中把通用寄存器(AX、BX、CX、DX、SI、DI、SP 和 BP)、指令指针寄存器(IP)和标志寄存器(FLAGS)从 16 位扩充成 32 位，其寄存器符号分别为：EAX、EBX、ECX、EDX、ESI、EDI、ESP 和 EBP。这些通用寄存器的低 16 位与先前 CPU 中的通用寄存器作用相一致，对其存取不会影响高 16 位的数据。

段寄存器(ES、CS、SS 和 DS)的长度没有变化，不过增加了 2 个 16 位的段寄存器：FS 和 GS。

3. 简述各通用寄存器的主要功能。

**答：**

通用寄存器可用于暂存数据，也可参与算术逻辑运算，并保存运算结果。除此之外，它们还各自具有一些规定的特殊功能。

（1）数据寄存器主要用来保存操作数和运算结果等信息，从而节省读取操作数所需占用总线和访问存储器的时间。

（2）变址和指针寄存器主要用于存放存储单元在段内的偏移量，用它们可表达多种访问内存的寻址方式，如寄存器间接寻址方式、寄存器相对寻址方式、基址加变址寻址方式

和相对基址加变址寻址方式等。变址和指针寄存器为使用不同的地址格式访问内存提供了极大的方便。

寄存器的主要用途如题表 2.1 所示。

题表 2.1 通用寄存器的主要用途

| 寄存器名称 | 寄存器符号 | 主 要 用 途 |
|---|---|---|
| 累加器 | EAX、AX、AL | 乘、除、I/O、快速的算术运算，中间结果的缓存 |
| 基地址寄存器 | EBX、BX | 在数据段中作基地址指针 |
| 计数器 | ECX、CX、CL | 循环计数器、重复操作计算器、移位操作计数器 |
| 数据寄存器 | EDX、DX | 字的乘、除运算，间接的输入输出 |
| 源变址寄存器 | ESI、SI | 存储器指针、串指令中的源操作数指针 |
| 源目标变址寄存器 | EDI、DI | 存储器指针、串指令中的目的操作数指针 |
| 基地址指针 | EBP、BP | 存储器指针、存取堆栈的指针 |
| 栈顶指针 | ESP、SP | 堆栈的栈顶指针 |

4．简述各段寄存器所指段的含义。

**答：**

16 位处理器有 4 个段寄存器：

（1）CS——代码段寄存器(Code Segment Register)；

（2）DS——数据段寄存器(Data Segment Register)；

（3）ES——附加段寄存器(Extra Segment Register)；

（4）SS——堆栈段寄存器(Stack Segment Register)。

32 位处理器共有 6 个 16 位段寄存器：CS、DS、ES、SS、FS 和 GS，前 4 个与 16 位处理器内的段寄存器完全相同，后 2 个是新增加的段寄存器，所以，在 32 位处理器环境下开发的程序可同时访问 6 个段。

5．IP / EIP 寄存器的用途是什么？它是“计算机组成原理”课中的什么概念在 CPU 中的具体体现？

**答：**

IP 和 EIP 分别是 16 位和 32 位的寄存器，它们存放着下次将要执行的指令在代码段的偏移量。它们是“计算机组成原理”中的指定“下条指令”地址的指令地址寄存器在 CPU 中的具体体现。

6．在标志寄存器中，用于反映运算结果属性的标志位有哪些？它们每一位所表示的含义是什么？在 Debug 环境下，用什么符号来表示？

**答：**

在标志寄存器中，用于反映运算结果属性的标志位有 6 个：CF、PF、AF、ZF、SF 和 OF。它们的含义和其值在 DEBUG 中相应的符号如题表 2.2 所示。

题表 2.2　标志寄存器中反映运算结果属性的标志位的含义及其值在 DEBUG 中相应的符号

| 标　志　位 | 作　　用 | DEBUG符号 | |
|---|---|---|---|
| | | 置位 | 复位 |
| CF | 反映运算是否产生进位或借位 | CY | NC |
| PF | 反映运算结果中“1”的个数的奇偶性 | PE | PO |
| AF | 在字操作时，低字节向高字节进位或借位；<br>在字节操作时，低4位向高4位进位或借位 | AC | NA |
| ZF | 反映运算结果是否为0 | ZR | NZ |
| SF | 反映运算结果的符号位，它与运算结果的最高位相同 | NG | PL |
| OF | 反映有符号数算术运算所得结果是否溢出 | OV | NV |

7．在标志寄存器中，用于反映 CPU 状态的标志位有哪些？它们每一位所表示的含义是什么？在 Debug 环境下用什么符号来表示？

**答：**

在标志寄存器中，用于反映 CPU 状态的标志位有 3 个：DF、IF 和 TF。它们的含义及其值在 DEBUG 中相应的符号如题表 2.3 所示。

题表 2.3　标志寄存器中反映 CPU 状态的标志位的含义及其值在 DEBUG 中相应的符号

| 标　志　位 | 作　　用 | DEBUG符号 | |
|---|---|---|---|
| | | 置位 | 复位 |
| DF | 用来决定在串操作指令执行时有关指针寄存器发生调整的方向 | DN | UP |
| IF | 用来决定CPU是否响应其外部的可屏蔽中断发出的中断请求 | EI | DI |
| TF | 当陷阱标志TF被置为1时，CPU进入单步执行方式，即每执行一条指令，产生一个单步中断请求。其主要用于程序的调试 | | |

8．填空

1）16 位/32 位 CPU 在取指令时，分别需要用到的寄存器有____和____、____和____；

2）访问堆栈段栈顶单元要由____段寄存器和____寄存器组合来寻址，访问堆栈段中的单元要由____段寄存器和____寄存器组合来寻址；

3）用基指针(BP)寻址存储器，则其用到的默认段寄存器是____；

4）访问数据段中的单元要由____段寄存器和____寄存器组合来寻址；

5）在 16 位/32 位微机系统中，程序最多能同时使用____数据段和____数据段。

**答：**

1）CS　IP　CS　EIP

2）SS　SP　SS　BP

3）SS

4）DS　BX(DI、SI)

5）2个 4个

9. 假设用以下寄存器组合来访问存储单元，试分别求出它们所访问单元的物理地址。

1）DS=1000H 和 DI=2000H；

2）DS=2000H 和 SI=1002H；

3）SS=2300H 和 BP=3200H；

4）DS=0A000H 和 BX=1000H；

5）SS=2900H 和 SP=3A00H。

**答：**

根据“物理地址=段地址×16+偏移量”的计算方法得出下列结果：

1）12000H；2）21002H；3）26200H；4）0A1000H；5）2CA00H。

10. 在实方式下，一个20位的物理地址最多可有____个段值和偏移量来表示。

**答：**

$2^{12}$

11. 在默认情况下，哪些寄存器可用于读取堆栈段的数据？

**答：**

在默认情况下，寄存器BP可用于读取堆栈段的数据。

12. 按组画出下列数值在内存中的存储形式(假设同组分配在连续存储单元中)：

1）字节值：1、2、‘1’、‘A’、‘a’和0；

2）字值：12H、6502H、-1和0；

3）双字值：-765H、12345678H和0。

**答：**

1）存储形式如题图2.1所示。

| … | 01H | 02H | 31H | 41H | 61H | 00H | … |
|---|---|---|---|---|---|---|---|

题图2.1 存储形式

2）存储形式如题图2.2所示。

| … | 12H | 00H | 02H | 65H | 0FFH | 0FFH | 00H | 00H | … |
|---|---|---|---|---|---|---|---|---|---|

题图2.2 存储形式

3）存储形式如题图2.3所示。

| … | 03H | 0FDH | 0FFH | 0FFH | 78H | 56H | 34H | 12H | 00H | 00H | 00H | 00H | … |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

题图2.3 存储形式

# 第三章　操作数的寻址方式

1．访问内存单元的寻址方式有几种？它们具体是哪些？

**答：**

16 位微机系统有五种访问内存单元的寻址方式：直接寻址方式、寄存器间接寻址方式、寄存器相对寻址方式、基址加变址寻址方式、相对基址加变址寻址方式。

32 位微机系统中，除上述五种寻址方式外，还增加了一种 32 位地址访问内存的寻址方式。

2．指出下列各种操作数的寻址方式：

（1）[BX]　　（2）SI

（3）435H　　（4）[BP+DI+123]

（5）[23]　　（6）data　　(data 是一个内存变量名)

（7）[DI+32]　　（8）[BX+SI]

（9）[EAX+90]　　（10）[BP+4]

**答：**

（1）寄存器间接寻址方式　　（2）寄存器寻址方式

（3）立即数寻址方式　　（4）相对基址加变址寻址方式

（5）直接寻址方式　　（6）直接寻址方式

（7）寄存器相对寻址方式　　（8）基址加变址寻址方式

（9）32 位地址的寻址方式　　（10）寄存器相对寻址方式

3．哪些寄存器的值可用于表示内存单元的偏移量？

**答：**

可用于表示内存单元的偏移量的寄存器为：BX、SI、DI 和 BP。

4．判断下列操作数的寻址方式的正确性，对正确的指出其寻址方式，错误的说明其错误原因：

（1）[AX]　　（2）[EAX]

（3）BP　　（4）[SI+DI]

（5）DS　　（6）BH

（7）[BX+BP+32]　　（8）[BL+44]

（9）[CX+90]　　（10）EDX

（11）BX+90H　　（12）[DX]

（13）SI[100H] （14）[BX*4]

（15）[EAX+EBX*6] （16）[DX+90H]

答：

（1）错误，AX 不能用于访问内存单元。

（2）正确，32 位地址的寻址方式。

（3）正确，寄存器寻址方式。

（4）错误，两个变址寄存器 SI 和 DI 不能同时用于内存单元的有效地址。

（5）正确，寄存器寻址方式。

（6）正确，寄存器寻址方式。

（7）错误，两基址寄存器 BX 和 BP 不能同时用于内存单元的有效地址。

（8）错误，寄存器 BL 不能用于表示内存单元的有效地址。

（9）错误，寄存器 CX 不能用于表示内存单元的有效地址。

（10）正确，寄存器寻址方式。

（11）错误，无此寻址方式或非法的表达式。

（12）错误，寄存器 DX 不能用于表示内存单元的有效地址。

（13）错误，书写方式不正确。

（14）错误，16 位寄存器不能乘比例因子。

（15）错误，32 位地址寻址方式的比例因子错。

（16）错误，寄存器 DX 不能用于表示内存单元的有效地址。

5．已知寄存器 EBX、DI 和 BP 的值分别为 12345H、0FFF0H 和 42H，试分别计算下列各操作数的有效地址：

（1）[BX] （2）[DI+123H]

（3）[BP+DI] （4）[BX+DI+200H]

（5）[1234H] （6）[EBX*2+345H]

答：

（1）2345H （2）113H （3）32H （4）2535H （5）1234H （6）249CFH

6．指出下列各寻址方式所使用的段寄存器。

（1）[SI+34H] （2）[456H]

（3）ES:[BP+DI] （4）[BX+DI+200H]

（5）[BP+1234H] （6）FS:[EBX*2+345H]

答：

（1）DS （2）DS （3）ES （4）DS （5）SS （6）FS

# 第四章　标识符和表达式

1．判断下列标识符的合法性。

（1）code　　（2）Data　　（3）ABCDH

（4）1abcH　　（5）eax　　（6）???01

（7）A?@　　（8）www@zsu　　（9）www@zsu.edu.cn

（10）AX$DX　　（11）_BX　　（12）CX

答：

（1）合法　　（2）合法　　（3）合法　　（4）不合法

（5）不合法　　（6）合法　　（7）合法　　（8）合法

（9）不合法　　（10）合法　　（11）合法　　（12）不合法

2．按下面要求写出相应的数据定义语句（未指定变量名的，可任意指定或默认）。

（1）定义一个字节区域，第一个字节的值为 20，其后跟 20 个初值为 0 的连续字节；

（2）定义一个以 0 为结束符的字符串，其初值为：The course is easy；

（3）定义一个以“$”为结束符的字符串，该串中含有换行符和回车符；

（4）定义 100 个字，其初值为 0；

（5）从一个偶地址开始定义一个字变量 word；

（6）先定义一个字符串 String，其偏移量指定为 10，其初值为“ABCDEF”，然后再定义 3 个字类型的数组 Data，要求 String 和 Data 的偏移量相同；

（7）定义一个字符串 String，其初值为“12345678”，再定义 4 个字类型的数组 Data，要求 String 和 Data 共享同一片内存单元。

答：

（1）
```
DB    20, 20 DUP(0)
```

（2）
```
DB    "The course is easy", 0
```

（3）
```
DB    0D, 0A, "$"
```

（4）
```
DW    100 DUP(0)
```

（5）
```
EVEN
word    DW ?
```

（6）
```
ORG 10
String  DB "ABCDEF"
ORG 10
Data    DW 3 dup (?)
```

（7）
```
String  DB  "12345678"
ORG  $-8
Data  DW 4 dup (?)
```

3．把下列 C 语言的变量定义语句改写成与之等价的汇编语言语句。

（1）char　ch1, ch2[3], ch3[ ]="Are you ready? ", ch4='Q';

（2）int w1, w2[5], w3=0x1234, w4[ ]={90, 89, 432, 344};

（3）long int d1=1234, d2, d3[10], d4[ ]={1, 2, 3, 4, 5, 6};

**答：**

（1）
```
ch1   db   ?
ch2   db   3 dup (?)
ch3   db   "Are you ready?", 0
ch4   db   "Q"
```

（2）
```
w1    dw   ?
w2    dw   5 dup (?)
w3    dw   1234h
w4    dw   90, 89, 432, 344
```

（3）
```
d1   dd   1234
d2   dd   ?
d3   dd   10 dup(?)
d4   dd   1, 2, 3, 4, 5, 6
```

4. 假设第 3 题中 3 组变量在内存中是连续存放的，且变量 ch1 的偏移量是 20H，试求出其他每个变量的偏移量。

**答：**

如题表 4.1 所示。

**题表 4.1 三组变量在内存中的偏移量**

| 变 量 | 偏 移 量 | 变 量 | 偏 移 量 | 变 量 | 偏 移 量 |
|---|---|---|---|---|---|
| ch1 | 20H | w1 | 34H | d1 | 4AH |
| ch2 | 21H | w2 | 36H | d2 | 4EH |
| ch3 | 24H | w3 | 40H | d3 | 52H |
| ch4 | 33H | w4 | 42H | d4 | 66H |

5. 按下列要求分别写出汇编语言的结构定义，如果没有说明结构分量的类型，请根据日常习惯自行指定其数据类型（可使用已有的结构定义）。

（1）定义日期结构类型 Date，其包括：Year(年)、Month(月)和 Day(日)；

（2）定义时间结构类型 Time，其包括：Hour(小时)、Minute(分)、Second(秒)和 Centsec (百分之一秒)；

（3）定义结构 Peason，其包括：Name(姓名：字符，长度为 8)、Birth(出生日期：Date)，Stature(身高：以厘米为单位的整数)、Address(地址：30 个字符)。

**答：**

（1）
```
Date    struc
        Year     dw   ?
        Month    db   ?
        Day      db   ?
```

（2）
```
Time    struc
        Hour     db   ?
        Minute   db   ?
        Second   db   ?
```

```
    Date    ends                                    Centsec db  ?
(3) Person  struc                           Time    ends
            Name    db   8 dup(?)
            Birth   Date < >
            Stature db   ?
            Address db   30 dup(?)
    Person  ends
```

6. 试求出第5题中各结构中字段的偏移量。

答：

```
          结构定义                                  偏移量
(1) Date    struc
            Year    dw  ?                          0
            Month   db  ?                          2
            Day     db  ?                          3
    Date    ends
(2) Time    struc
            Hour    db  ?                          0
            Minute  db  ?                          1
            Second  db  ?                          2
            Centsec db  ?                          3
    Time    ends
(3) Person  struc
            Name    db   8 dup(?)                  0
            Birth   Date < >                       8
            Stature db   ?                         12
            Address db   30 dup(?)                 13
    Person  ends
```

7. 把下列C语言的结构或结构变量的定义改写为与之等价的汇编语言定义语句。

```
(1) struct  ScreenBuffer {
            char    cl;             //字符
            char    attr;           //字符属性
            } Buffer[2000];
(2) struct  FILE {
            int     level, flags;
            char    fd, hold;
            int     bsize, istemp, token;
            };
(3) struct  CELLREC {
            char  attrib;
```

```
union {
  char  text[201];
  long  value;
  struct {
         long  fvalue;
         char  formula[201];
         } f;
           } v;
};
```

答:

(1)
```
ScreenBuffer    struc
                cl      db  ?
                attr    db  ?
ScreenBuffer    ends
Buffer          ScreenBuffer    2000 dup(?)
```

(2)
```
FILE    struc
        level   dw  ?
        flags   dw  ?
        fd      db  ?
        hold    db  ?
        bsize   dw  ?
        istemp  dw  ?
        token   dw  ?
FILE    ends
```

(3)
```
struc_f     struc
            fvalue  dd  ?
            formula db 201 dup(?)
struc_f     ends
union_v     union
            text    db 201  dup(?)
            value   dd      ?
            f       struc_f < >
union_v     ends
CELLREC     struc
            attrib  db      ?
            v       union_v < >
CELLREC     ends
```

8. 按下列要求分别写出汇编语言的记录定义。

(1) 定义标志寄存器低 8 位标志位，其相应位用其标志位的符号来命名，未用部分的

命名自行确定；

（2）定义一个压缩型的 BCD 码类型；

（3）把一个字节分成 3 部分：0~2 位、3~5 位和 6~7 位，每部分可独立存取，它们的初值分别为 4、5 和 2；

（4）把一个字分割成：0~1 位、2~6 位、7~12 位，其中第一部分的初值为 3，第三部分的初值为 25。

**答：**

```
(1) flags   record  SF:1, ZF:1, empty1:1, AF:1, empty2:1, PF:1, empty3:1,
    CF:1
(2) BCD     record  high:4, low:4
(3) record1 record  s1:2=2, s2:3=5, s3:3=4
(4) record2 record  s4:6=25, s5:5, s6:2=3
```

9. 试求出第 8 题中各记录中字段的宽度和屏蔽属性值。

**答：**

| | 字段名 | 宽度 | 屏蔽属性值 |
|---|---|---|---|
| (1) | SF | 1 | 80h |
| | ZF | 1 | 40h |
| | empty1 | 1 | 20h |
| | AF | 1 | 10h |
| | empty2 | 1 | 08h |
| | PF | 1 | 04h |
| | empty3 | 1 | 02h |
| | CF | 1 | 01h |
| (2) | high | 4 | 0f0h |
| | low | 4 | 0fh |
| (3) | s1 | 2 | 0c0h |
| | s2 | 3 | 38h |
| | s3 | 3 | 07h |
| (4) | s4 | 6 | 1f80h |
| | s5 | 5 | 007ch |
| | s6 | 2 | 0003h |

10. 假设变量 B1 的偏移量为 12H，试求出以下变量的属性 OFFSET、TYPE、LENGTH 和 SIZE 的值。

```
B1      DB      "The programming language is easy", 0
W1      DW      10 DUP (1, 2), 1234
D1      DD      20 DUP (0)
P1      PEASON  <>, <>          ;结构 PEASON 的定义见 4.4.2
```

```
U1      UNION1    3 DUP (< >)      ;联合 UNION1 的定义见 4.4.3
```

**答：**

```
          OFFSET  TYPE    LENGTH  SIZE
B1         12H      1       1      1
W1         33H      2       10     20
D1         5DH      4       20     80
P1         0ADH     15      1      15
U1         0CBH     2       3      6
```

11. 试用 3 种方法定义字节变量（或字段名）B1 和字变量（或字段名）W1，它们共享 20 个存储单元。

**答：**

方式 1：

```
u1  union
    B1  db  20 dup(?)
    W1  db  10 dup(?)
u1  ends
```

方式 2：

```
W1  equ this word
B1  db  20 dup(?)
```

方式 3：

```
B1  db  20 dup(?)
org $-20
W1  db  10 dup(?)
```

12. 强制属性操作符会改变变量的定义属性吗？在 C 语言中存在类似功能的语句吗？若存在，请举例说明。

**答：**

在指令中使用强制属性操作符 PTR 后，不管其后的地址表达式原数据类型是什么，在本指令中都以 PTR 前面的类型为准。该强制属性只在本指令有效，是一种临时性的属性，它不会改变原内存单元的定义属性。

在 C 语言中，也存在类型强制方式。例如：

```
int I=1;
long K;
K = (long)I;
```

C 语言的类型强制具有一定的限制，不能像汇编语言一样自由地进行类型间转换。

13. 假设有第 10 题中的变量定义，试写出符合要求的地址表达式。

（1）取从 B1 开始的字母“p”和第一个“m”；

（2）访问从 W1 开始的第 3 个值为 2 字单元（从序数 1 开始计数，第 1、第 2、……）的存储单元；

（3）访问从 D1 开始的第 10 个双字单元；

（4）访问从 P1 开始的第 2 个结构中的字段 NO；

（5）访问从 U1 开始的第 3 个联合中的字段 W1。

**答：**

（1）[b1+4], [b1+10]；（2）[w1+10]；（3）[d1+36]；（4）[p1+15]；（5）[u1+4]；

14. 把下列 C 语言程序中的说明性语句改写成等价的汇编语言语句。

```
#define  N  200
struct buffer {
       char len1, len2, buff[N];
       } buff;
int Data[N];
```

**答：**

```
N       equ 200
buffer  struct
        len1    db  ?
        len2    db  ?
        buff    db  N   dup(?)
buffer  ends
buff    buffer  < >
Data    dw      N   dup(?)
```

15. 简述数值表达式与地址表达式的主要区别。

**答：**

数值表达式是在汇编过程中能够由汇编程序计算出来的数值型表达式，它的各组成部分在汇编时就能完全确定。它通常是一些常量的运算组合。地址表达式是计算存储单元地址的表达式，它可由标号、变量名和由括号括起来的基址或变址寄存器组成。其计算结果表示一个存储单元的地址，而不是该存储单元的值。地址表达式的结果不能在汇编时确定，只有在程序执行过程中才能确定。

16. 计算下列各数值表达式的值。

（1）12H AND 78H　　（2）1234 MOD 3 + 23H

（3）5467H XOR 12FFH　　（4）'A' OR 20H

（5）10 LT 9　　（6）4567 / 13 + 10H

（7）1234H+89H SHL 1　　（8）8910H SHR 2 AND 506H

**答：**

（1）0010H　（2）0024h　（3）4698h　（4）0061H

（5）0FFFFH　（6）016FH　（7）1346H　（8）0004H

# 第五章　微机 CPU 的指令系统

1．书写正确的 MOV 指令，需要符合哪些规定？在 32 位 CPU 中，增加了什么指令来实现不同数据类型之间的数据传送？

**答：**

书写 MOV 指令需要符合以下几条规定：

（1）两个操作数的数据类型要相同，要同为 8 位、16 位或 32 位；如：MOV　BL, AX 等是不正确的；

（2）两个操作数不能同时为段寄存器，如：MOV　ES, DS 等；

（3）代码段寄存器 CS 不能为目的操作数，但可作为源操作数，如：指令 MOV　CS, AX 等不正确，但指令 MOV　AX, CS 等是正确的；

（4）立即数不能直接传给段寄存器，如：MOV　DS, 100H 等；

（5）立即数不能作为目的操作数，如：MOV　100H, AX 等；

（6）指令指针 IP，不能作为 MOV 指令的操作数；

（7）两个操作数不能同时为存储单元，如：MOV　VARA, VARB 等，其中 VARA 和 VARB 是同数据类型的内存变量。

对于规定（2）、（4）和（7），可以用通用寄存器作为中转来达到最终目的。

在 80386 及其以后的 CPU 中，增加了一组新的指令 MOVSX 和 MOVZX，其作用是数据的传送/填充，它可把位数少的源操作数传送给位数多的目的操作数，多出的部分按指令的规定进行填充。

2．简述指令 MOV BX, 1234H、MOV BX, [1234]和 MOV BX, offset [1234H]的功能和它们之间的区别。

**答：**

（1）MOV BX, 1234H 的作用是将通用寄存器 BX 的当前值设为 1234H；

（2）MOV BX, [1234]的作用是取出内存地址 1234H 的当前值赋予通用寄存器 BX；

（3）MOV BX, offset [1234H]的作用是取内存地址 1234H 的偏移量，而地址[1234H]的偏移量就是 1234H，所以，指令的执行效果就是把 1234H 赋予寄存器 BX。

3．简述指令 MOV BX, Data 和 MOV BX, offset Data 之间的区别。

**答：**

（1）MOV BX, Data 的作用是将 Data 的值赋予通用寄存器 BX；

（2）MOV BX, offset Data 的作用是将 Data 的偏移地址赋予寄存器 BX。

4. 选择指令实现下列要求（$OPN_1$←$OPN_2$ 的含义：把操作数 $OPN_2$ 的值传送给 $OPN_1$）。

（1）EBX←EDX　　（2）BX←CL

（3）SI←BX　　（4）DS←AX

（5）AL←BH　　（6）AL←12H

（7）AX←13AH　　（8）CX←0CDH (假设 0CDH 为无符号数)

（9）SI←1000H　　（10）EBX←1200A2H

答：

```
(1) MOV  EBX, EDX          (2) MOV  SX, BX, CL
(3) MOV  SI, BX            (4) MOV  DS, AX
(5) MOV  AL, BH            (6) MOV  AL, 12H
(7) MOV  AX, 13AH          (8) MOV  CX, 0CDH
(9) MOV  SI, 1000H         (10) MOV  EBX, 1200A2H
```

5. PUSHA 指令能把哪些寄存器压栈？它们进栈的顺序是什么？可用什么指令把这些寄存器依次弹出堆栈？

答：

PUSHA 指令能把寄存器 AX、CX、DX、BX、SP、BP、SI 和 DI 依次压入堆栈，可以使用 POPA 指令将这些寄存器依次弹出堆栈。

6. PUSHAD 指令完成什么操作？与它作用相反的操作是什么？

答：

PUSHAD 指令能把寄存器 EAX、ECX、EDX、EBX、ESP、EBP、ESI 和 EDI 依次压入堆栈，它与 POPAD 的作用相反。

7. 在 16 位/32 位 CPU 中，标志位寄存器的内容进栈和出栈的指令是什么？

答：

PUSHF/PUSHFD(Push Flags onto Stack)：16 位/32 位标志寄存器进栈；

POPF/POPFD(Pop Flags off Stack)：16 位/32 位标志寄存器出栈。

8. 写出下列指令序列中每条指令的执行结果，请在 Debug 环境下验证之，并注意各标志位的变化情况。

```
MOV   BX, 23ABH
ADD   BL, 0ACH
MOV   AX, 23F5H
ADD   BH, AL
SBB   BX, AX
ADC   AX, 12H
SUB   BH, -9
```

答：

| 指令 | 执行结果 | 标志位 |
|---|---|---|
| MOV BX, 23ABH | BX=23ABH | … |
| ADD BL, 0ACH | BX=2357H | OV、PL、AC、PO、CY |
| MOV AX, 23F5H | AX=23F5H | … |
| ADD BH, AL | BX=1857H | NV、NA、PE |
| SBB BX, AX | BX=F461H | NG、PO |
| ADC AX, 12H | AX=2408H | PL、NC |
| SUB BH, -9 | BX=FD61H | NG、AC、CY |

9．假定(DS)=1123H，(SS)=1400H，(BX)=0200H，(BP)=1050H，(DI)=0400H，(SI)=0500H，LIST 的偏移量为 250H，试确定下面各指令访问内存单元的地址。

（1）MOV AL, [1234H]　　（2）MOV AX, [BX]

（3）MOV [DI], AL　　（4）MOV [2000H], AL

（5）MOV AL, [BP+DI]　　（6）MOV CX, [DI]

（7）MOV EDX, [BP]　　（8）MOV LIST[SI], EDX

（9）MOV CL, LIST[BX+SI]　　（10）MOV CH, [BX+SI]

（11）MOV EAX, [BP+200H]　　（12）MOV AL, [BP+SI+200H]

（13）MOV AL, [SI-0100H]　　（14）MOV BX, [BX+4]

答：

（1）内存单元地址：1234H　　（2）内存单元地址：0200H

（3）内存单元地址：0400H　　（4）内存单元地址：2000H

（5）内存单元地址：1450H　　（6）内存单元地址：0400H

（7）内存单元地址：1050H　　（8）内存单元地址：0750H

（9）内存单元地址：0950H　　（10）内存单元地址：0700H

（11）内存单元地址：1250H　　（12）内存单元地址：1750H

（13）内存单元地址：0400H　　（14）内存单元地址：0204H

10．用一条指令实现把(BX)和(SI)之和传送给 CX。

答：

LEA CX, [BX+SI]

11．试比较下列 3 组指令的功能。

（1）LDS SI, [DI]

（2）MOV SI, [DI]
MOV DS, [DI+2]

（3）MOV SI, [DI+2]
MOV SI, [DI]

答：

（1）相当于 MOV SI, [DI]

MOV DS，[DI+2]

（2）功能与（1）组相同。

（3）第一条指令 MOV DS，[DI+2]改变了 DS 值，因此随后的 MOV SI，[DI]的执行结果可能会不同于（1）、（2）两组指令。

12．标志位 CF 有着重要的作用，请简述之，并至少写出 3 种把 CF 置 0 的方法。

**答：**

进位标志 CF 主要用来反映运算是否产生进位或借位。如果运算结果的最高位产生了一个进位或借位，那么其值为 1，否则其值为 0。

将 CF 置 0 的方法有：

① CLC

② XOR AX, AX

③ SUB AX, AX

13．写一个计算 DL 3 次方的指令序列，假设幂不超过 16 位二进制数。

**答：**

```
MOV   AL, DL
MUL   DL
MOV   BX, AX
MOV   AL, DL
CBW
MUL   BX
```

计算结果存放在 AX 中

14．写出一个指令序列，要求实现：用 CL 中的数除 BL 中的数，然后商乘 2，最后的结果存入 DX 中。

**答：**

```
MOV   AL, BL
CBW
DIV   CL
CBW
SHL   AX, 2
MOV   DX, AX
```

15．按下列要求编写指令序列。

（1）清除 DH 中的低 3 位而不改变其他位，结果存入 BH 中；

（2）把 DI 中的高 5 位置 1 而不改变其他位；

（3）把 AX 中的 0~3 位置 1，7~9 位取反，13~15 位置 0；

（4）检查 BX 中的第 2、5 和 9 位中是否有 1 位为 1；

（5）检查 CX 中的第 1、6 和 11 位是否同时为 1；

（6）检查 AX 中的第 0、2、9 和 13 位中是否有 1 位为 0；

（7）检查 DX 中的第 1、4、11 和 14 位是否同时为 0。

**答：**

```
(1) AND   DH, 0F8H
    MOV   BH, DH
(2) OR    DI, 0F800H
(3) OR    AX, 000FH
    XOR   AX, 0380H
    AND   AX, 0E000H
(4) AND   BX, 0224H      ;如果运算后 BX 的值不为 0，则说明至少有 1 位为 1。
(5) AND   CX, 0842H
    XOR   CX, 0842H      ;如果运算后 CX 的值为 0，则说明 3 位全部为 1。
(6) OR    AX, 0DDFAH     ;如果运算后 AX 的值不为 0FFFFH，则说明至少有 1 位为 0。
(7) AND   DX, 4812H      ;如果运算后 DX 的值为 0，说明 4 位同时为 0。
```

16. 简述指令 AND 和 TEST、NOT 和 NEG 之间的区别。

**答：**

TEST 检测位指令是把两个操作数进行逻辑“与”操作，并根据运算结果设置相应的标志位。TEST 指令并不保存该运算结果，所以不会改变指令中的操作数，这一点与 AND 指令不同。

NEG 是求补指令，指令的功能相当于“操作数=0 -操作数”，即改变操作数的正、负号。NOT 是逻辑非指令，其作用是将操作数按位取反。

17. 选择适当的指令实现下列功能。

（1）右移 DI 3 位，并把 0 移入最高位；

（2）把 AL 左移 1 位，使 0 移入最低 1 位；

（3）AL 循环左移 3 位；

（4）EDX 带进位位循环右移 4 位；

（5）DX 右移 6 位，且移位前后的正、负性质不变。

**答：**

```
(1) SHR  DI, 3          (2) SHL   AL,  1
(3) ROL  AL, 3          (4) RCR   EDX, 4
(5) SAR DX, 6
```

18. 判断下列指令的正确性，若某指令是错误的，请说明其错误原因。

（1）MOV BL, CX　　（2）MOV DS, SS

（3）MOV　[BX], [DI]　　（4）MOV　AL, [BX][SI]
（5）MOV　ES, AL　　（6）MOV　DS, DX
（7）MOV　CS, AX　　（8）MOV　BX, CS
（9）MOV　DS, 1230H　　（10）MOVSX　DS, AL
（11）XCHG　BX, 3　　（12）POP　CS
（13）MOV　IP, SI　　（14）PUSH　CS
（15）PUSH　BL　　（16）MOV　[SP], BX
（17）MOV　AX, BX+3　　（18）MOV　AX, [BX+3]
（19）MOV　BX, [BX]　　（20）MOV　BH, [BL]
（21）XCHG　ES, AX　　（22）LEA　AX, [BX+SI]
（23）MUL　10H　　（24）IMUL　DX, 10H
（25）DIV　10　　（26）IDIV　DX, 10H
（27）SHL　AX, CX　　（28）SHR　BX, CH
（29）ROL　BX, 20　　（30）RCR　AX, CL
（31）CMP　AX,1234H　　（32）CMP　12H, CL
（33）JCXZ　next　　（34）JEBXZ　next　(假设标号 next 已存在)

**答：**

（1）错误，寄存器类型（位数）不匹配。
（2）错误，不能直接将段寄存器的值赋予另外一个段寄存器。
（3）错误，两个操作数不能同为内存单元。
（4）正确。
（5）错误，寄存器类型（位数）不匹配。
（6）正确。
（7）错误，不能修改 CS 值。
（8）正确。
（9）错误，不能用立即数给段寄存器赋值。
（10）只能用于 80386 及以上的 CPU。
（11）错误，不能使用立即数。
（12）错误，不能修改 CS 值。
（13）错误，不能修改 IP 值。
（14）正确。
（15）错误，PUSH 的操作数不能是 8 位寄存器。
（16）错误，[ ]中必须使用 BX、BP、SI 和 DI，不能使用 SP。
（17）错误，语法错、错误的表达式。

（18）正确。

（19）正确。

（20）错误，8 位寄存器不能用于取地址。

（21）错误，寄存器不能是段寄存器。

（22）正确。

（23）错误，单操作数乘法不能使用立即数。

（24）只能在 80286 及以上的 CPU 使用。

（25）错误，DIV 的除数不能使用立即数。

（26）错误，无此指令格式，IDIV 的除数不能使用立即数。

（27）错误，第二操作数可以是 CL，不能是 CX。

（28）错误，第二操作数可以是 CL，不能是 CH。

（29）在 8086 中错误，在 8086 以上的 CPU 中可以。

（30）正确。

（31）正确。

（32）错误，立即数不能作为目的操作数。

（33）正确。

（34）错误，无此指令。

19．方向标志 DF 的作用是什么？用于设置或消除该标志位的指令是什么？

**答：**

方向标志 DF 用来决定在串操作指令执行时有关指针寄存器发生调整的方向。

清方向位指令 CLD(Clear Direction Flag)：DF←0。

置方向位指令 STD(Set Direction Flag)：DF←1。

20．串指令用 DI 和 SI 寻址哪些内存段中的内存数据？

**答：**

串指令用 DI 寻址 ES 段的内存数据，用 SI 寻址 DS 段的内存数据。

21．段间转移和段内转移之间的区别是什么？

**答：**

转移可以是一个短(short)转移(偏移量在[–128, 127]范围内)，近(near)转移(偏移量在[–32K, 32K]范围内)或远(far)转移(在不同的代码段之间转移)。

短转移和近转移是段内转移，JMP 指令只把目标指令位置的偏移量赋给指令指针寄存器 IP，从而实现转移功能。但远转移是段间转移，JMP 指令不仅会改变指令指针寄存器 IP 的值，而且还会改变代码段寄存器 CS 的值。

22．SCASB 指令的作用是什么？叙述指令 REPE　SCASB 指令所完成的功能。

**答：**

该指令是用指针 ES:DI 所指向字节、字或双字的值与相应的 AL、AX 或 EAX 的值相减，用所得到的差来设置有关标志位。与此同时，变址寄存器 DI 还将根据标志位 DF 进行增减。

REPE　SCASB 是相等重复前缀指令，该重复前缀指令的执行步骤如下：

（1）判断条件：(CX)≠0 且 ZF=1；

（2）如果条件不成立，则结束重复操作，执行程序中的下一条指令；

（3）如果条件成立(CX) = (CX) -1 (不影响有关标志位)，并执行其后的字符串操作指令，在该指令执行完后，再转到步骤（1）。

23．指令 REPNE　SCASB 结束执行的条件是什么？

**答：**

REPNE SCASB 是不等重复前缀指令，该重复前缀指令的执行步骤如下：

该重复前缀指令的执行步骤如下：

（1）判断条件：(CX)≠0 且 ZF=0；

（2）如果条件不成立，则结束重复操作，执行程序中的下一条指令；

（3）如果条件成立(CX) = (CX)-1 (不影响有关标志位)，并执行其后的字符串操作指令，在该指令执行完后，再转到步骤(1)。

24．REP 前缀的作用是什么？能用指令 REP LODSB 读取 DS:SI 所指内存中的每个字符来进行处理吗？若不能，请说明原因。

**答：**

REP 是重复前缀指令，其作用是重复其后字符串操作指令，重复次数由 CX 来决定。用 REP LODSB 虽然可以读取 DS:SI 所指内存中的一连串字符，但是由于其他指令根本无法处理取出的所有数据。指令结束后留在 AL 中的仅仅是最后一个取出字符的 ASCII 码。

25．编写指令序列，在字符串 LIST 中查寻字符“B”，若找到，转向 Found；否则，转向 NotFound，假设该字符串含有 300 个字符。

**答：**

```
CLD                         ; 设置 DF=0，每次执行字符串指令后 SI 增加
MOV   AX, SEG LIST
MOV   DS, AX
LEA   SI, LIST              ; 用 DS:SI 来指向字符串的首地址
MOV   CX, 300               ; 重复次数
MOV   AL,   42H             ; AL="B"
REPZ  SCASB
JZ    Found
```

```
JMP NotFound
```

26. 编写指令序列，把 Source 存储区域中的 12 个字节传送到 Dest 存储区域内。

**答：**

```
CLD                          ; 设置 DF=0，每次执行字符串指令后 SI、DI 增加
MOV   AX, SEG Source
MOV   DS, AX
LEA   SI,  Source            ; 用 DS:SI 来指向 Source 的首地址
MOV   AX, SEG Dest
MOV   ES, AX
LEA   DI,  Dest              ; 用 ES:DI 来指向 Dest 的首地址
MOV   CX, 12
REP   MOVSB
```

27. 设计一个短指令序列，将 32 位数 AX:BX 中的 8 位 BCD 与 CX:DX 中的 8 位 BCD 相加，并把所得结果存入 CX:DX 中。

**答：**

```
PUSH  AX
MOV   AX, BX
ADD   AL, DL
DAA
MOV   DL, AL
MOV   AL, BH
ADC   AL, DH
DAA
MOV   DH, AL
POP   AX
ADC   AL, CL
DAA
MOV   CL, AL
MOV   AL, AH
ADC   AL, CH
DAA
MOV   CH, AL
```

28. 有符号数比较后，用什么样的条件转移指令实现转移？无符号数比较后，用什么样的条件转移指令实现转移？

**答：**

（1）无符号数的条件转移指令(Jumps Based on Unsigned (Logic) Data)，如题表 5.1 所示。

**题表 5.1　无符号数的条件转移指令**

| 指令的助忆符 | 检测的转移条件 | 功 能 描 述 |
|---|---|---|
| JE/JZ | ZF=1 | 若相等或为 0，则转移 |
| JNE/JNZ | ZF=0 | 若不等或不为 0，则转移 |
| JA/JNBE | CF=0 and ZF=0 | 若“高于”或“不低于”、“等于”，则转移 |
| JAE/JNB | CF=0 | 若“高于”、“等于”或“不低于”，则转移 |
| JB/JNAE | CF=1 | 若“低于”或“不高于”、“等于”，则转移 |
| JBE/JNA | CF=1 or AF=1 | 若“低于”、“等于”或“不高于”，则转移 |

（2）有符号数的条件转移指令(Jumps Based on Signed (Arithmetic) Data)，如题表 5.2 所示。

**题表 5.2　有符号数的条件转移指令**

| 指令的助忆符 | 检测的转移条件 | 功 能 描 述 |
|---|---|---|
| JE/JZ | ZF=1 | 若相等或为 0，则转移 |
| JNE/JNZ | ZF=0 | 若不等或不为 0，则转移 |
| JG/JNLE | ZF=0 and SF=OF | 若“大于”或“不小于”、“等于”，则转移 |
| JGE/JNL | SF=OF | 若“大于”、“等于”或“不小于”，则转移 |
| JL/JNGE | SF≠OF | 若“小于”或“不大于”、“等于”，则转移 |
| JLE/JNG | ZF=1 or SF≠OF | 若“小于”、“等于”或“不大于”，则转移 |

# 第六章　程序的基本结构

1．简述 ASSUME 伪指令的作用，用该语句说明的段寄存器是否需要对其赋段地址的初值？

**答：**

在汇编语言源程序中可以定义多个段，每个段都要与一个段寄存器建立一种对应关系。建立这种对应关系的说明语句格式如下：

ASSUME　段寄存器名:段名[, 段寄存器名:段名，… …]

ASSUME 说明语句只起说明作用，它不会对段寄存器赋值，所以必须对有关段寄存器赋初值。

2．用两种方法定义堆栈段，并给出相应的初始化语句或说明。

**答：**

方法 1：

```
stack1  segment
        db      256 dup(?)   ;256 是堆栈的长度，可根据需要进行改变
        top     label  word
stack1 ends
```

初始化语句如下：

```
assume  ss:stack1              ; 可在代码段的段指定语句中一起说明
cli                            ; 禁止响应可屏蔽中断
mov   ax, stack1
mov   ss, ax
mov   sp, offset top           ; 给堆栈段的栈顶寄存器 sp 赋初值
sti                            ; 恢复响应可屏蔽中断
```

方法 2：

```
stack1  segment  stack         ; 定义一个堆栈段，其段名为 stack1
   db   256 dup(?)
stack1  ends
```

上述段定义说明了该段是堆栈段，系统会自动把段寄存器 SS 和栈顶寄存器 SP 与该堆栈段建立相应的关系，并设置其初值，而不用在代码段对段寄存器和栈顶寄存器 SP 赋值。

3．把下列 C 语句的语句改写成功能相同的汇编语言程序片段(其中：变量都为整型变量)。

（1）h = (key & 0XFF00) >> 8;

（2）k = (k + '1' – 0xabcd) / 56;

（3）for (i = s = 0; i < 100; i++) s += data[i] * 2;

（4）for (s = 0, i = 100; i > 0; i–) s += i * 2;

**答：**

（1）
```
        key   dw  ?
        h     dw  ?
        ...
        mov   ax, key
        and   ax, 0FF00H
        mov   cl, 8
        sar   ax, 8
        mov   h,  ax
```

（2）
```
        k     dw  ?
        ...
        mov   ax, k
        mov   bh, 0
        mov   bi, 31h
        add   ax, bx
        sub   ax, 0abcdh
        mov   dx, 0
        mov   bx, 56
        div   bx
        mov   k,  ax
```

（3）
```
        I     dw ?
        s     dw ?
        data  dw100 dup(?)
        ...
        mov   I,  0
        mov   s,  0
        mov   si, I
        mov   ax, s
    next1:
        cmp   si, 100
        jge   next2
        mov   bx, data[si]
        sal   bx, 1
        add   ax, bx
        inc   si
        jmp   next1
```

```
    next2:
        mov   s, ax
(4)     s     dw 0
        I     dw 0
        ...
        mov   s, 0
        mov   I, 100
        mov   cx, I
        mov   ax, s
    next1:
        mov   bx, cx
        sal   bx, 1
        add   ax, bx
        loop  next1
        mov   s,  ax
```

4. 把下列 C 语言的语句改写成等价的汇编语言程序段(不考虑运算过程中的溢出)。

```
If (a<1 || b/4 > 10 && c%8==5) {
  a = 20+b++;
  c <<= 2;
}
else {a = 21-(++c);
  b--;
}
```

其中：变量 a、b 和 c 都是有符号的整型(int)变量。

**答：**

```
        _a    dw ?
        _b    dw ?
        _c    dw ?
        ...
        mov   bx, _b
        mov   cl, 2
        sar   bx, cl                    ; 计算 b/4，结果存在 bx 中
        mov   cx, _c
        and   cx, 0007h                 ; 计算 c%8，结果存在 cx 中
        cmp   bx, 10
        jbe   next1
        cmp   cx, 5
        jne   next1
```

```
next2:                                  ; (a<1 || b/4 > 10 && c%8==5)计算为真
    mov   bx, _b
    mov   ax, 20
    add   ax, bx
    mov   _a, ax
    inc   bx
    mov   _b, bx
    mov   cx, _c
    sal   cx, 1
    sal   cx, 1
    mov   _c, cx
    jmp   next3
next1:
    mov   ax, _a
    cmp   ax, 1
    jb    next2
    mov   cx, _c              ; (a<1 || b/4 > 10 && c%8==5)计算为假
    inc   cx
    mov   _c, cx
    mov   ax, 21
    sub   ax, cx
    mov   _a, ax
    mov   bx, _b
    dec   bx
    mov   _b, bx
next3:
```

5. 假设内存单元中有 3 个字 a、b 和 c，编写一个程序，它可判断它们能否构成一个三角形，若能，CF 为 1，否则，CF 为 0。

**答：**

判断三条边是否能构成一个三角形的原则是：任意一边的边长都不得大于等于另外两边的边长之和。对应的C语言语句如下：

```
If ((a+b)>c && (a+c>b) && (b+c)>a) {
  //能够构成三角形;
}
else {
  //不能构成三角形;
}
```

相应的汇编语言代码如下：

```
    _a    dw ?
    _b    dw ?
    _c    dw ?

    mov   ax, _a
    add   ax, _b
    cmp   ax, _c
    jbe   failure
    mov   ax, _a
    add   ax, _c
    cmp   ax, _b
    jbe   failure
    mov   ax, _b
    add   ax, _c
    cmp   ax, _a
    jbe   failure
    stc                         ; 设置CF=1
    jmp   end_of_proc
failure:
    clc                         ; 设置CF=0
end_of_proc:
```

6. 假设有 3 个无符号字存放在以 Buffer 为开始的缓冲区中，编写一个程序把它们从低到高排序。

**答：**

```
    Buffer  dw  3 dup (?)

    mov   ax, Buffer
    mov   bx, Buffer[2]
    mov   cx, buffer[4]
    cmp   ax, bx
    jbe   next1
    xchg  ax, bx
next1:
    cmp   bx, cx
    jbe   next2
    xchg  bx, cx
next2:
    cmp   ax, bx
```

```
        jbe   next3
        xchg  ax, bx
    next3:
        mov   Buffer, ax
        mov   Buffer[2], bx
        mov   Buffer[4], cx
```

7. 编写一个程序，它把一位十六进制数转化成相应的数字字符或大写字母。

**答：**

假设这个十六进制数存储在寄存器 AL 中，该数的值必须小于 0x10。该数被转换后输出到寄存器 BL 中。

```
        mov   bl, al
        and   bl, 0fh
        cmp   bl, 0ah
        jb    next1
        add   bl, 7h
    next1:
        add   bl, 30h
```

8. 编写一个程序，它把一个合法的十六进制字符转化成相应的数值。

**答：**

这个字符必须是'0'~'9'、'A'~'F'、'a'~'f'中的某一个，否则它就不是合法的十六进制字符。

假设该字符存放在 al 中，并且已经通过了合法性判断。

该字符被转换后输出结果将存放在 bl 中。

```
        mov   bl, al
        cmp   bl, 'a'
        jb    next1
        sub   bl, 'a'-10
        add   bl, 9
        jmp   next2
    next1:
        cmp   bl, 'A'
        jb    next3
        sub   bl, 'A'-10
        jmp   next2
    next3:
        sub   bl, 30h
    next2:
```

9. 编写一个程序，它可统计 32 位数 DX:AX 中二进制位是 1 的位数。

**答：**

以下程序将统计 DX:AX 中 1 的个数，并将统计结果存入 BL 中。

```
        mov   BL, 0
        mov   cx, 16
    again:
        rol   ax, 1
        adc   bl, 0
        rol   dx, 1
        adc   bl, 0
        loop  again
```

10. 编写一个程序，它把 CH 和 CL 中的二进制位依次交叉存入 AX 中。

**答：**

下面的程序完成该任务：将 CH 的 0~7 位的值依次存入 AX 的 1 位、3 位、5 位、…、15 位，并将 CL 的 0~7 位的值依次存入 AX 的 0 位、2 位、4 位、…、14 位。

```
        Mov   BX, CX
        Mov   CX, 8
    again:
        rol   bh, 1
        rcl   ax, 1
        rol   bl, 1
        rcl   ax, 1
        loop  again
```

11. 编写一个程序，求出从内存单元 1000:0000 开始的 1 024 个字的 32 位累加和，并把该值存入程序中的变量 data 中。

**答：**

```
        data   dd  ?

        mov    bx, 1000
        mov    es, bx
        mov    di, 0
        mov    ax, 0
        mov    dx, 0
        mov    cx, 1024
    next1:
        add    ax, es:[di]
        adc    dx, 0
        add    di, 2
```

```
    loop   next1
    mov    word ptr data,    ax
    mov    word ptr data[2], dx
```

12. 假设从变量 Buff 开始存放了 200 个字，编写一个程序统计出其正数、0 和负数的个数，并把它们分别存入 N1、N2 和 N3 中。

**答：**

```
    Buff  dw  200   dup(?)
    N1    dw  0
    N2    dw  0
    N3    dw  0
    ...
    mov   cx, 200
    mov   si, 0
next4:
    mov   ax, word ptr Buff[si]
    add   si, 2
    cmp   ax, 0
    je    next1
    jg    next2
    mov   bx, offset N3
    inc   word ptr [bx]
    jmp   next3
next1:
    mov   bx, offset N2
    inc   word ptr [bx]
    jmp   next3
next2:
    mov   bx, offset N1
    inc   word ptr [bx]
next3:
    loop  next4
```

13. 用双重循环把下三角乘法表存入从 product 开始的 45 个字节中。

**答：**

```
    product db 45 dup(?)
    ...
    mov   bx, offset product
    mov   dh, 1
next2:
    mov   dl, 1
```

```
next1:
    mov   al, dh
    mul   dl
    mov   byte ptr [bx], al
    inc   bx
    inc   dl
    cmp   dl, dh
    jbe   next1
    inc   dh
    cmp   dh, 9
    jbe   next2
```

14. 表示源程序结束的伪指令是什么？在其后所编写的指令能被汇编吗？

**答：**

表示源程序结束的伪指令是 END，在其后编写的任何内容都不会被汇编程序处理。因此，通常情况下伪指令 END 是汇编语言源程序的最后一条语句。

15. 汇编语言程序一定会从代码段的第一条指令开始执行吗？如果不是，如何指定程序的入口地址？

**答：**

汇编语言程序可以不从代码段的第一条指令开始执行。

在伪指令 END 后面可附带一个在程序中已定义的标号，该标号指明程序的入口地址。

16. 编写一个程序，它把字符串 String 两端的空格删除(字符串以 0 结束)。

**答：**

假设 String 定义如下：

```
String  db 100 dup (?), 0
s     dw?
e     dw?

;寻找字符串开头的非空格位置
    mov   si, 0
next2:
    cmp   String[si], 20h
    jne   next1
    inc   si
    jmp   next2
next1:
    mov   s, si                ; s 指向字符串的第一个非空字符
next4:
```

```
    cmp   String[si], 0       ;寻找字符串尾
    je    next3
    inc   si
    jmp   next4
next3:
    cmp   String[si-1], 20h ;寻找字符串结尾的非空格位置
    jne   next5
    dec   si
    jmp   next3
next5:
    mov   e, si               ; e 指向字符串的第一个非空字符
    ;移动字符串
    mov   cx, e
    sub   cx, s
    mov   si, s
    mov   di, 0
    mov   ax, ds
    mov   es, ax
    cld
    rep   movsb
    mov   [di], 0
```

17. 编写一个程序，它把字符串 String 中的数字字符删除(字符串以 0 结束)。

**答：**

假设 String 定义为：

```
    String  db 100 dup (?), 0
    ...
    ;从字符串头开始逐个字符检查
    mov   si, offset String
    mov   di, si
    mov   ax, ds
    mov   es, ax
    cld
next2:
    cmp   BYTE ptr [si], 30h
    jb    next1
    cmp   BYTE ptr [si], 39h
    ja    next1
    inc   si
```

```
    jmp   next2
next1:
    movsb
    cmp   BYTE ptr [si-1], 0
    jne   next2
```

18. 编写一个程序，它把 AX 的值以二进制字符串的形式存放在以 buff 开始的缓冲区中。

答：

buff 定义如下：

```
    buff  db  16 dup (?)
    ...
    mov   cx, 16
    mov   bx, offset buff
next2:
    sal   ax, 1
    jc    set_1
    mov   BYTE ptr [bx], 30h
    jmp   next1
set_1:
    mov   BYTE ptr [bx], 31h
next1:
    inc   bx
    loop  next2
```

19. 假设有一个以 Data 为首地址的字缓冲区，其缓冲区的大小在第一个字中，试编写一个程序，求出该缓冲区数值的最大值和最小值，并分别存入 DI 和 SI 中。

答：

Data 定义如下：

```
    _Size  EQU  100
    Data   dw _size, _size dup (?)   ;设置缓冲区大小为 100 个字
    ...
    mov    cx, Data
    mov    bx, offset Data
    mov    si, [bx]
    mov    di, [bx]
next2:
    add    bx, 2
    mov    ax, [bx]
    cmp    ax, si
```

```
    jge    next1
    mov    si, ax
next1:
    cmp    ax, di
    jle    next3
    mov    di, ax
next3:
    loop    next2
```

20. 编写一个程序片段，把 32 位数 DX:AX 向左移 6 位。

**答：**

```
mov   bx, ax
and   bx, 0fc00h
mov   cl, 10
shr   bx, cl
mov   cl, 6
shl   dx, cl
or    dx, bx
mov   cl, 6
shl   ax, cl
```

# 第七章　子程序和库

1．汇编语言的子程序是如何定义的？所使用的关键字有哪些？

**答：**

定义子程序的一般格式如下：

```
子程序名    PROC    [NEAR | FAR]
            …                    ;子程序体
子程序名    ENDP
```

所使用的关键字包括：PROC、 ENDP、NEAR 和 FAR。

2．为了编写具有良好风格的子程序，一般需要书写哪些重要的说明性信息？

**答：**

在编写子程序时，除了要考虑实现子程序功能的方法，还要养成书写子程序说明信息的好习惯。说明信息一般包括以下几方面内容：

（1）功能描述

（2）入口和出口参数

（3）所用寄存器　可选项，最好采用寄存器的保护和恢复方法，使之使用透明化；

（4）所用额外存储单元　可选项，可以减少为子程序定义自己的局部变量；

（5）子程序所采用的算法　可选项，如果算法简单，可以不写；

（6）调用时的注意事项　可选项，尽量避免除入口参数外还有其他的要求；

（7）子程序的编写者　可选项，为将来的维护提供信息；

（8）子程序的编写日期　可选项，用于确定程序是否是最新版本。

这些说明性信息虽然不是子程序功能的一部分，但其他程序员可通过它们对该子程序的整体信息有一个较清晰认识，为准确地调用它们提供直接的帮助，与此同时，也为实现子程序的共享提供了必要的资料。

3．简述调用指令 CALL 和转移指令 JMP 之间的主要区别。

**答：**

CALL 指令是一种特殊的转移操作。当调用一个子程序时，程序的执行顺序被改变，CPU 将转而执行子程序中的指令序列，在这方面，CALL 指令含有转移指令的功能。

JMP 转移指令是一种“一去不复返”的操作，而当子程序执行结束后，还要求 CPU 能转而执行 CALL 指令后续的指令，它是一种“有去有回”的操作。

4．简述段内和段间子程序调用指令 CALL 的主要区别。

**答：**

子程序的调用指令分为近(near)调用和远(far)调用。近调用即段内子程序调用，远调用即段间子程序调用。

如果被调用子程序的属性是近的，那么，CALL 指令将产生一个近调用，它把该指令之后地址的偏移量(用一个字来表示)压栈，把被调用子程序入口地址的偏移量送给指令指针寄存器 IP 即可实现执行程序的转移。

如果被调用子程序的属性是远的，那么，CALL 指令将产生一个远调用。这时，调用指令不仅要把该指令之后地址的偏移量压进栈，而且也要把段寄存器 CS 的值压进栈。在此之后，再把被调用子程序入口地址的偏移量和段值分别送给 IP 和 CS，这样完成了子程序的远调用操作。

5．子程序返回指令 RET 的功能能否用 JMP 指令来模拟，若可以，请用段内子程序的返回加以说明；否则，说明理由。

**答：**

子程序返回指令 RET 的功能可以使用 JMP 指令来模拟。

以段内子程序的返回为例，以下两条语句可以模拟指令 RET：

```
POP     AX
JMP     AX
```

6．子程序返回指令“RET 6”的功能是返回数值“6”给调用程序吗？若不是，其作用是什么？

**答：**

子程序返回指令“RET 6”的功能并不是返回数值“6”给调用程序。这个“6”是一个立即数，表示在得到返回地址之后，SP 的值还要另外加 6。

7．在子程序中要使其所用寄存器对调用者是透明的，请举例说明达到此目的的方法。

**答：**

在子程序中要使其所用寄存器对调用者透明，需要在子程序中保存和恢复寄存器内容。主要方法是：在子程序的开始把它所用到的寄存器值压进栈，在返回前，再把它们弹出栈。这样编写的好处是该子程序可以被任何其他程序调用。在调用指令前，不需要保存寄存器，在调用指令后，也无需恢复寄存器。

下面的子程序修改了寄存器 AX 的值，要通过堆栈来保护 AX 的原始值。

```
sample  proc
        push    ax
        …
        mov     ax, 100
        …
```

```
        pop ax
        ret
sample  endp
```

8. 编写子程序实现下列功能，参数的传递方式可自行决定(假设所有变量都是字类型)。

(1) ABS($x$)=|$x$|

(2) $f(x)=3x^2+5x-8$

(3) strlen(String) //返回字符串 String 的长度，该字符串以 0 结束

(4) 判断 3 条边 a、b 和 c 能否构成三角形，若能，CF 为 1；否则，CF 为 0。

**答：**

(1)

```
abs   proc          ;参数 x 由寄存器 ax 传递
      cmp   ax, 0
      jge   ok
      not   ax
      inc   ax
ok:
      ret
abs   endp
```

该子程序在处理绝对值最大的一个负数-32 768 时会发生溢出，因为字变量最大正数为+32 767。如果 AX 的值为-32 768，则子程序返回时 AX 的值依然保持-32 768。

(2) 当 $x$ 是个字变量时，函数 $f(x)=3x^2+5x-8$ 的计算结果可能会超过双字变量的表达能力，即发生溢出。为了简化程序，假定 $x$ 足够小，使得 $f(x)$的值可以用字变量保存。

子程序 func_f 用于计算该函数，参数 $x$ 由寄存器 AX 传递，计算结果存储在 BX 中。

```
func_f  proc
        push  cx
        push  dx
        mov   bx, ax
        imul  bl
        mov   cx, 3
        imul  cx
        xchg  bx, ax
        mov   cl, 5
        imul  cl
        add   bx, ax
        sub   bx, 8
        pop   dx
        pop   cx
```

```
        ret
func_f  endp
```

（3）子程序 strlen 用于计算字符串的长度，假定长度不超过字变量的表达范围。返回值通过寄存器 AX 传递。

```
strlen  proc
        push    bx
        mov     ax, 0
        lea     bx, offset string
next1:
        cmp     byte ptr [bx] ,  0
        je      string_end
        inc     bx
        inc     ax
        jmp     next1
string_end:
        pop     bx
        ret
strlen  endp
```

（4）答：略

9. 编写计算 CX 值 3 次方的子程序，结果也存入 CX(不考虑溢出)。

答：

```
tri_x   proc
        push    ax
        push    dx
        mov     ax, cx
        imul    cx
        imul    cx
        mov     cx, ax
        pop     dx
        pop     ax
        ret
tri_x   endp
```

10. 编写计算表达式(AX)=(DI)×(SI) / 100H 的子程序(不考虑溢出)。

答：

```
sample  proc
        push    bx
        push    dx
        mov     ax,   di
        imul    si
```

```
        mov     bx, 100h
        idiv    bx
        pop     dx
        pop     bx
        ret
sample  endp
```

11. 编写计算(EAX) = (EAX) + (EBX) + (ECX) + (EDX)的子程序。在计算过程中，若产生进位，则 EDI=1，否则，EDI=0。

**答：**

```
sample  proc
        mov     edi,0
        add     eax,ebx
        adc     edi
        add     eax,ecx
        adc     edi
        add     eax,edx
        adc     edi
        cmp     edi,0
        je      ok
        mov     edi,1
ok:
        ret
sample  endp
```

12. 在 MASM 6.x 编程环境下，简述子程序定义中 USES 说明语句的作用。

**答：**

USES 是子程序定义中保护寄存器说明子句，其格式为：

USES 寄存器列表

该说明子句要求汇编程序为其生成保护和恢复寄存器的指令序列，即在进入子程序执行指令之前，把寄存器列表中的寄存器压进堆栈；在结束子程序执行时，把先前压进堆栈的寄存器弹出，以达到保护寄存器的目的。

寄存器列表列举出在子程序中需要保护的寄存器名，即在子程序开始时需要把内容进栈的寄存器名。若有多个寄存器名，则在寄存器名之间要用“空格”来分开。

13. 调用子程序指令 CALL 和调用伪指令 INVOKE 的主要区别是什么？

**答：**

INVOKE 伪指令是调用基于堆栈的子程序方法，它把所有参数压栈，子程序结束时，又把参数自动弹出堆栈。在参数传递时，汇编程序将根据子程序的原型进行数据类型检查。

若需要进行参数类型转换的话，汇编程序则会自动生成一段代码来满足其数据类型转换的要求。

而 CALL 指令只负责调用子程序，并不对参数进行处理。

14．在子程序的完全定义形式中，其语言类型所起的作用是什么？可用哪两种方法来设定子程序的语言类型？

**答：**

子程序语言类型(Language Type)可以是任何一种有效的程序设计语句类型，由它来告诉汇编程序将使用什么样的标识符命名风格、子程序的调用和返回约定。该语言类型说明可使汇编语言程序与其他语言程序达到共享的目的。

定义子程序时，程序员可先用.MODEL 伪指令，或使用<langtype>参数来说明本子程序所使用的程序设计语言类型。

15．如何指定子程序的传递参数是动态的，对动态参数有哪些规定？

**答：**

子程序参数可以使用数据类型 VARARG。VARARG 数据类型允许向子程序传递“个数”不定的参数，其参数之间要用逗号“,”来分开。

若参数表中含有 VARARG 说明的参数，那么，该参数一定是该子程序的最后一个参数。其规定隐含地说明了在参数表中只能有一个用 VARARG 说明的参数。

当子程序的语言类型是 C、SYSCALL 和 STDCALL 时，在其参数表中才能使用 VARARG 数据类型的参数。

16．编写一个子程序，其功能是把其所有参数数值之和存入 AX 中，每个参数都为 16 位二进制数，但个数不定。

**答：**

参数的个数 parmcount 作为第一个参数传递给子程序，其后跟着参数表 parmvalues。

```
addup   proc near c uses cx si, parmcount:word, parmvalues:vararg
        xor     ax, ax
        xor     si, si
        mov     cx, parmcount
        .repeat
          add   ax, parmvalues[si]
          inc   si
          inc   si
        .untilcxz
        ret
addup   endp
```

17．子程序的参数传递有传值和传地址之分，在汇编语言中，如何实现传地址？请举

例说明。

**答：**

地址有近地址和远地址之分。近地址有16位，远地址有32位。子程序参数传递可以实现地址传递，近地址可以使用一个16位寄存器传递，远地址可以使用两个16位寄存器传递。

例如STUDENT是一个结构类型，ZhangShan是这个结构的一个实例。子程序CALC功能是计算单个学生成绩，它的输入参数是学生实例的近地址，通过AX寄存器传入。

```
STUDENT    STRUCT
           ...
STUDENT    ENDS
ZhangShan  STUDENT   < >

CALC       PROC
           ...
CALC       ENDP

;调用程序如下
MOV    AX, OFFSET ZhangShan
CALL   CALC
```

18. 在高级语言中，子程序可定义其局部变量，在汇编语言中，能定义其局部变量吗？若能，请举例说明。

**答：**

汇编语言可以通过伪指令LOCAL定义子程序的局部变量。

在子程序中，若说明了某个局部变量，则子程序体中的指令就可使用该局部变量。汇编程序会把对它的引用转换成用指针寄存器BP来访问其在堆栈中的实际存储单元。

局部变量的作用域与高级语言中局部变量的作用域相一致，即局部变量只能在当前子程序中使用，离开该子程序，它们就不能再被引用。但在局部变量的命名规则上有所不同，高级语言中的局部变量可与外层变量同名，而汇编语言中的局部变量不能与其他任何变量同名，否则，在汇编时，将会给出“重定义”(Symbol Redefinition)的错误信息。

例如：

```
LOCAL  data[20]:BYTE, num:WORD
```

在上例的说明中，定义了两个局部变量：data和num。前者是字节类型，并有20个元素，后者是字类型，只有其自身1个元素。

19. 使用.REPEAT和.UNTLL语句编写一个子程序，其功能是从DS:SI开始的内存单

元中查找是否有与 AL 相同的数值。若找不到，CF=0；否则，CF=1，且 SI 指向所找到位置。假设查找区域以 0 为结束标志。

**答：**

```
search  proc
      .repeat
         .break.if  [si]==al
         inc si
       .until  [si]==0
       .if  [si]==al
         stc
       .else
         clc
       .endif
       ret
search  endp
```

20. 用.REPEAT 和.UNTILCXZ 语句编写一个子程序，用以在内存缓冲区中填入 00H。内存缓冲区的首址及长度分别由 DS:SI 和 CX 来确定。

**答：**

```
fill    proc
         .repeat
            mov  byte ptr[si], 0
            inc  si
         .untilcxz
         ret
fill    endp
```

21. 用.WHILE 和.ENDW 语句实现 19、20 题的功能。

**答：**

将 19 题用.WHILE 和.ENDW 语句实现：

```
search  proc
        .while  byte ptr [si]!=0
          .break.if  [si]==al
          inc    si
        .endw
        .if  [si]==al
           stc
        .else
```

```
            clc
          .endif
          ret
search    endp
```

将 20 题用.WHILE 和.ENDW 语句实现：

```
fill   proc
         .while   cx!=0
         mov      byte ptr[si],0
         inc      si
         dec      cx
       .endw
       ret
fill   endp
```

22. 在程序模块中，伪指令 PUBLIC 和 EXTRN 的作用是什么？

**答：**

为了清晰地描述各模块之间的联系，汇编语言提供了两条伪指令 PUBLIC 和 EXTRN，它们的作用有点像 C/C++语言说明变量、过程和函数是“全局的”或“外部的”。

伪指令 PUBLIC 用来说明当前模块中哪些标识符是能被其他模块引用的公共标识符。说明的一般格式如下：

PUBLIC 标识符$_1$, 标识符$_2$, ……

其中：“标识符”可以是变量名、过程名和程序标号，各标识符之间要用逗号分开。

伪指令 EXTRN 是用来说明：在当前模块所使用的标识符中，哪些标识符是已在其他模块中被定义为指定类型的标识符。如果当前模块使用了其他模块的标识符，而对它又不加以说明的话，那么，在汇编时，汇编程序将会给出下列出错信息：

error nnnnn: undefined symbol : XXXXXX

其中：“nnnnn”是错误号，“XXXXXX”是当前模块中没有定义的标识符。

23. 在 C 语言程序中，存在关键字 EXTERN，它与汇编语言中 EXTRN 的作用相同吗？

**答：**

C 语言程序中的关键字 EXTERN 与汇编语言中的 EXTRN 的思想是相似的，但作用有所不同。C 语言程序中的关键字 EXTERN 不但可以用来描述子程序，而且可以用来描述变量。而汇编语言的 EXTRN 只用于描述子程序。

24. 如何创建和维护自己的子程序库？使用子程序库有何好处？

**答：**

宏汇编 MASM 系统提供了建立库文件的命令文件 LIB.EXE。

程序员在编写源程序时，通常采用模块化的思想来组织源程序：把各类不同的子程序

分别编写在不同的模块中，在各模块中说明所用到的、在其他模块中已定义的子程序，或说明本模块所定义的子程序可被其他模块调用。这样组织后，就可以分别汇编它们而得到其相应的目标文件，在有了这些目标文件后，程序员就可用不同的方法来生成最终的可执行文件。

用库文件来存储子程序可生成较短的可执行文件。

# 第八章　输入输出和中断

1．简述中断和子程序调用之间的主要区别。

**答：**

中断和子程序调用在实现方面的主要差异：

（1）子程序调用一定是程序员在编写源程序时事先安排好的，是可知的，而中断是由中断源根据自身的需要产生的，是不可预见的(用指令 INT 引起的中断除外)；

（2）子程序调用是用 CALL 指令来实现的，但没有调用中断的指令，只有发出中断请求的事件(指令 INT 是发出内部中断信号，而不要理解为调用中断服务程序)；

（3）子程序的返回指令是 RET，而中断服务程序的返回指令是 IRET/IRETD；

（4）在通常情况下，子程序是由应用系统的开发者编写的，而中断服务程序是由系统软件设计者编写的。

2．为什么要区分 IRET 指令与 RET 指令?

**答：**

响应中断的过程是先保护现场和返回地址的，然后再进入中断响应程序的运行。现场包括通用寄存器和标志位寄存器的内容，而调用子程序一般由用户自己来保护特定的寄存器，通常不保护标志位寄存器。由于保护内容的不同，在返回时，从堆栈中弹出的内容也就不同。所以，为了区别返回指令的功能，需要分别设置中断返回指令 IRET 指令和子程序返回 RET 指令。

3．编写一个子程序，它可显示以 0 结尾的字符串。子程序的入口参数 DS:DX 为待输出字符串的首地址。

**答：**

```
Display   proc   uses    ax  bx       ; 子句 uses 在 masm6.x 中有效，
next:
          mov    bx, dx
          mov    dl, byte ptr [bx]
          cmp    dl, 0
          je     finish
          mov    ah, 2
          int    21h
          inc    bx
          jmp    next
```

```
finish:
        ret
Display endp
```

4. 编写一个子程序，用来读入一个键，并在屏幕上按十六进制的形式显示按键的扩展 ASCII 码，如果按键为普通字符，则不显示。

答：

```
code1   segment"code"
        assume  cs:code1
Display proc uses ax            ; 以十六进制的形式显示寄存器 DL 低 4 位二进制
        and   dl, ofh
        cmp   dl, 9
        jb    below1
        add   dl, 7h
below1:
        add   dl, 30h
        mov   ah, 2
        int   21h
        ret
Display endp
DispKey proc uses ax cx dx
        mov   ah, 0
        int   16h               ; 从键盘读入字符
        cmp   al, 0             ; 如果为普通字符则子程序结束
        jne   over
        mov   dl, ah
        mov   cl, 4
        shr   dl, cl
        call  Display           ; 按十六进制显示高 4 位二进制
        mov   dl, ah
        call  Display           ; 按十六进制显示低 4 位二进制
over:
        ret
DispKey endp
start:
        call  DispKey           ; 调用子程序
        mov   ax, 4c00h         ; 程序结束
        int   21h
code1   ends
        end   start
```

5. 编写一个程序，在屏幕的右下角闪烁显示编程者自己的姓名，显示颜色自定。

答：

```
Data1   segment
        name1   db  "Li Lei"
        dispmode db ?
Data1   ends
code1   segment
        assume  cs:code1, Ds:Data1
start:
        mov   ax, Data1          ;初始化数据段寄存器
        mov   ds, ax
        mov   ah, 0fh            ;保存当前显示模式
        int   10h
        mov   dispmode, al
        mov   ah, 0
        mov   al, 3              ;设置为 80×25 彩色文本显示模式
        int   10h
        mov   ax, ds
        mov   es, ax
        mov   bp, offset name1
        mov   dx, 1849h          ; 坐标（24，73）
        mov   cx, 6
        mov   bh, 0
        mov   bl, 9ah            ; 显示字符的颜色
        mov   ah, 13h
        mov   al, 0
        int   10h                ;在屏幕右下角显示姓名
        mov   ah, 0              ;等待键盘输入
        int   16h
        mov   ah, 0              ;恢复原先的显示模式
        mov   al, dispmode
        int   10h
        mov   ax, 4c00h          ;程序结束
        int   21h
code1   ends
        end start
```

6. 编写一个把屏幕上显示的字母经过大小写转换后再显示的程序。

答：

```
.model  small
.code
        .startup
```

```
        mov     ax, ob800h
        mov     ds, ax
        mov     cx, 2000              ; 屏幕上所有字母个数：25×80
        xor     bx, bx                ; 屏幕缓冲区中的第 0 个字节
again:
        mov     al, [bx]
        .if(al>='a')&&(al<='z')       ; Masm v6.x 支持，其他可能不支持这种写
                                        法，下同
          sub   al, 20h               ; 小写字母变为大写字母
        .elseif(al>='A')&&(al<='Z')
          add   al, 20h               ; 小写字母变为大写字母
        .endif
        mov     [bx], al              ; 把变化后的字母保存到显示缓冲区中
        and     bx, 2                 ; 移到下一个显示字母
        loop    again
        .exit
        end
```

7. 编写一个程序，它把屏幕上的数字改为蓝绿色背景的红字。

**答：**

```
.model  small
.code
        .startup
        mov     ax, 0b800h
        mov     ds, ax
        mov     cx,2000               ; 屏幕上所有字符个数：25×80
        xor     bx, bx                ; 屏幕缓冲区中的第 0 个字节
again:
        mov     ax, [bx]              ; al 是显示字符，ah 是字符的显示属性
        .if(al>='0')&&(al<='9')
          mov   ah, 34h               ; 改变数字字符的属性
        .endif
        mov     [bx], ax              ; 把变化后的字符保存到显示缓冲区中
        add      bx, 2                ; 移到下一个显示字符
        loop     again
        .exit
        end
```

8. 编写一个控制光标位置和形状的程序，该程序具有以下功能：

（1）可用光标移动键↑、↓、←和→来移动光标；

（2）当光标已在第 0 列，且按←键时，光标定在上一行的最后一列；若已在屏幕的左

上角，则光标不动，且给出响铃；按→键时的边界处理类似；

（3）当光标在第0行，且按↑键时，则光标不动，且给出响铃；按↓键时的边界处理类似；

（4）按Home或End键，则光标移到当前行的行首或行尾；

（5）若按下数字或字母键，则把该字符从当前位置依次显示到屏幕顶(在新位置显示当前字符时，原位置的符号被抹去)；

（6）按Esc键，程序结束。

**答：**

```
Code1   segment
        assume  cs:Code1, ds:Data1

;get current cursor position
;input: none
;output: dh—line number, dl—column number
GetCursorPosition   proc    uses ax bx cx
        mov   ah, 3
        mov   bh, 0
        int   10h
        ret
GetCursorPosition   endp
;set current cursor     position
;input: dh—line number, dl—column number
;output: none
SetCursorPosition   proc    uses ax bx cx
        mov   ah, 2
        mov   bh, 0
        int   10h
        ret
SetCursorPosition   endp
;move cursor to the head of line
;input: none
;output: none
HomeKey         proc
        call   GetCursorPosition
        mov    dl, 0
        call   SetCursorPosition
        ret
HomeKey         endp
```

```
;move cursor to the end of line
;input: none
;output: none
EndKey      proc
        call   GetCursorPosition
        mov    dl, 79
        call   SetCursorPosition
        ret
EndKey      endp
;Bell Ring
;input: none
;output: none
RingBell    proc    uses ax bx cx
        mov   ah, 2
        mov   dl, 7
        int   21h
        ret
RingBell        endp
;move cursor to next line
;input: none
;output: none
DownArrowKey   proc
        call   GetCursorPosition
        cmp    dh, 24
        jae    Bell1
        inc    dh
        call   SetCursorPosition
        ret
Bell1:
        call   RingBell
        ret
DownArrowKey   endp
;move cursor to previous line
;input: none
;output: none
UpArrowKey      proc
        call    GetCursorPosition
        cmp     dh, 0
        je      Bell2
```

```
        dec     dh
        call    SetCursorPosition
        ret
Bell2:
        call    RingBell
        ret
UpArrowKey      endp
;move cursor to left position
;input: none
;output: none
LeftArrowKey    proc
        call    GetCursorPosition
        cmp     dl, 0
        je      PreviousLine
        dec     dl
        call    SetCursorPosition
        ret
PreviousLine:
        cmp     dh, 0
        je      Bell3
        dec     dh
        mov     dl, 79
        call    SetCursorPosition
        ret
Bell3:
        call    RingBell
        ret
LeftArrowKey    endp
;move cursor to right position
;input: none
;output: none
RightArrowKey    proc
        call    GetCursorPosition
        cmp     dl, 79
        jae     NextLine
        inc     dl
        call    SetCursorPosition
        ret
```

```
NextLine:
        cmp     dh, 24
        je      Bell4
        inc     dh
        mov     dl, 0
        call    SetCursorPosition
        ret
Bell4:
        call    RingBell
        ret
RightArrowKey endp
;Key ascii character at the first line with the same column to cursor
;input: al—ascii character
;output: none
WriteKey        proc    uses ax bx cx
        call    GetCursorPosition
        push    dx
        mov     dh, 0
        call    SetCursorPosition
        mov     ah, 0ah
        mov     bx, 0
        mov     cx, 1
        int     10h
        pop     dx
        call    SetCursorPosition
        ret
WriteKey        endp
Start:
        mov     ax, Data1
        mov     ds, ax
NextKey:
        mov     ah, 0           ;wait for any Key be pressed
        int     16h
        cmp     ax, 11bh        ;check ESC
        je      Finsh
        cmp     ax, 4700h
        jne     Next1
        call    HomeKey
```

```
        jmp     NextKey
Next1:
        cmp     ax, 4f00h
        jne     Next2
        call    EndKey
        jmp     NextKey
Next2:
        cmp     ax, 5000h
        jne     Next3
        call    DownArrowKey
        jmp     NextKey
Next3:
        cmp     ax, 4b00h
        jne     Next4
        call    LeftArrowKey
        jmp     NextKey
Next4:
        cmp     ax, 4d00h
        jne     Next5
        call    RightArrowKey
        jmp     NextKey
Next5:
        cmp     ax, 4800h
        Jne     Next6
        call    UpArrowKey
        jmp     NextKey
Next6:
        .if(al>='0')&&(al<='9')||(al>='A')&&(al<='Z')||(al>='a')&&
        (al<='Z')
          call  WriteKey
          jmp   NextKey
        .endif
        jmp     NextKey
Finsh:
        Mov     ax, 4c00h
        int     21h
Code1   ends
```

9. 交换例 8.5 和 8.6 的画线方法，编写相应的子程序 VLine 和 HLine，并验证之。

**答：**

例 8.5 的画垂直线的子程序如下：

```
;draw a vertical line
;input: cx—column number, al—color
;output: none
VLine   proc    uses bx cx dies
        mov   bx, 0a000h
        mov   di, cx
        mov   cx, 200
NextLine:
        mov   es, bx
        mov   byte ptr es:[di], al
        add   bx, 20
        loop  NextLine
        ret
VLine   endp
```

例 8.6 的画水平线的子程序如下：

```
;draw a horizon line
;input: dx—line number, al—color
;output: none
Hline   proc  uses ax cx
        mov   cx, 0
next:
        mov   ah, 0ch
        int   10h
        inc   cx
        cmp   cx, 320
        jb    next
        endp
Hline   endp
```

10. 假设显示器的显示模式设定为 12H，编写实现下列功能的程序：

（1）在屏幕中间从上到下显示一条明亮的蓝色线，线宽为 1 个像素；

（2）在屏幕底下横向画一条绿色线，线宽为 2 个像素；

（3）在屏幕上垂直显示 16 种颜色，每种颜色宽 40 个像素；

（4）设定屏幕背景为白色，在屏幕中间画一条青色线，线宽为 10 个像素。

**答：**

（1）

```
Data1   Segment
        dispmode db ?
```

```
Data1   Ends
Code1   Segment
        assume  cs:Code1, ds:Data1
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ah, 0fh           ;save current display mode
        int   10h
        mov   dispmode, al
        mov   ah, 0
        mov   al, 12h
        int   10h
        mov   dx, 0
next:
        mov   ah, 0ch
        mov   al, 9             ;bright blue
        mov   cx, 320
        int   10h
        inc   dx
        cmp   dx, 480
        jb    next
        mov   ah, 0             ;wait for any key be pressed
        int   16h
        mov   ah, 0         ;restore display mode
        mov   al, dispmode
        int   10h
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

(2)

```
Data1   Segment
        dispmode db ?
Data1   Ends
Code1   Segment
        assume  cs:Code1, ds:Data1
Start:
        mov   ax, Data1
        mov   ds, ax
```

```
        mov   ah, 0fh           ;save current display mode
        int   10h
        mov   dispmode, al
        mov   ah, 0
        mov   al, 12h
        int   10h
        mov   dx, 478
nextl:
        mov   cx, 0
next:
        mov   ah, 0ch
        mov   al, 2             ;green
        int   10h
        inc   cx
        cmp   cx, 640
        jb    next
        inc   dx
        cmp   dx, 480
        jb    nextl
        mov   ah, 0             ;wait for any key be pressed
        int   16h
        mov   ah, 0             ;restore display mode
        mov   al, dispmode
        int   10h
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

(3)

```
Data1   Segment
        dispmode db ?
Data1   Ends
Code1   Segment
        assume  cs:Code1, ds:Data1
;Draw a vertical line
;input: cx—column number, al—color
DrawVLine   proc    uses dx bx ax
```

```
        push   dx
        push   bx
        push   ax
        mov    dx, 0
NextPoint:
        mov    ah, 0ch
        int    10h
        inc    dx
        cmp    dx, 480
        jb     NextPoint
        ret
DrawVLine   endp
Start:
        mov    ax, Data1
        mov    ds, ax
        mov    ah, 0fh          ;save current display mode
        int    10h
        mov    dispmode, al
        mov    ah, 0            ;set to 640×480×16 mode
        mov    al, 12h
        int    10h
        mov    al, 0
        mov    cx, 0
        mov    si, 0
next1:
        call   DrawVLine
        inc    cx
        cmp    cx, 640
        je     over
        inc    si
        cmp    si, 40
        jb     next1
        mov    si, 0
        inc    al
        jmp    next1
over:
        mov    ah, 0            ;wait for any key be pressed
        int    16h
```

```
        mov    ah, 0              ;restore display mode
        mov    al, dispmode
        int    10h
        mov    ax, 4c00h
        int    21h
Code1   Ends
        End Start
```

(4)

```
Data1   Segment
        dispmode db ?
Data1   Ends
Code1   Segment
        assume  cs:Code1, ds:Data1
;Draw a vertical line
;input: cx—column number, al—color
DrawVLine   proc    uses ax bx dx
        mov   dx, 0
NextPoint:
        mov   ah, 0ch
        int   10h
        inc   dx
        cmp   dx, 480
        jb    NextPoint
        ret
DrawVLine   endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ah, 0fh              ;save current display mode
        int   10h
        mov   dispmode, al
        mov   ah, 0                ;set to 640×480×16 mode
        mov   al, 12h
        int   10h
        mov   cx, 0
        mov   al, 15               ;white color
next1:
        call  DrawVLine
```

```
        inc   cx
        cmp   cx, 640
        jb    next1
        mov   cx, 315
        mov   al, 11
next2:
        call  DrawVLine
        inc   cx
        cmp   cx, 325
        jb    next2
        mov   ah, 0              ;wait for any key be pressed
        int   16h
        mov   ah, 0              ;restore display mode
        mov   al, dispmode
        int   10h
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

11. 对于256色，320×200的显示模式需要多少个字节的内存存放一屏信息?

**答:**

表达256种颜色需要一个字节。因此，一屏信息需要的内存字节数=320×200=64 000。

12. 为什么在显示数据时必须关闭鼠标?

**答:**

鼠标显示的过程中需要将鼠标的像素掩码与当前鼠标所处位置的像素信息进行“逻辑与”运算，还要将鼠标的光标掩码与“逻辑与”后的运算结果再进行“异或”操作。当鼠标移开时，还需要进行类似的反操作。如果在显示数据时不关闭鼠标，假如程序在鼠标位置放置数据，一旦鼠标移开，显示的结果就不再是原先的内容了。所以在显示数据时必须关闭鼠标。

13. 编写程序，检测计算机是否已安装了鼠标，并显示Yes/No来表示检测结果。

**答:**

```
Data1   Segment
        Yes   db "Yes!", 13, 10, '$'
        No    db "No!", 13, 10, '$'
Data1   Ends
Code1   Segment
        assume  cs:Code1, ds:Data1
```

```
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ax, 0
        int   33h
        cmp   ax, 0
        je    NotInstall
        mov   dx, offset Yes
        jmp   disp
NotInstall:
        mov   dx, offset No
disp:
        mov   ah, 9
        int   21h
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

14. 编写程序，显示鼠标的按键，若按左键，显示“Left”，若按右键，显示“Right”，按 Esc 键，程序结束。

答：

```
Data1   Segment
        NoMouse   db "Mouse isn't Installed!", 13, 10, '$'
        Left      db "Left", 13, 10, '$'
        Right     db "Right", 13, 10, '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ax, 0
        int   33h
        cmp   ax, 0
        jne   Installed
        mov   ah, 9
        mov   dx, offset NoMouse
        int   21h
```

```
        jmp   over
Installed:
        mov   ah, 1              ;read keyboard status
        int   16h
        jz    NoAnyKey
        cmp   ax, 11bh
        je    over               ;Esc key been pressed
NoAnyKey:
        mov   ax, 3
        int   33h
        test  bx, 1
        je    Next
        mov   ah, 9
        mov   dx, offset Left
        int   21h
Next:
        test  bx, 2
        je    Installed
        mov   ah, 9
        mov   dx, offset Right
        int   21h
        jmp   Installed
over:
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

15. 编写一个程序，在图形模式下，打开鼠标，并显示鼠标的位置。

**答：**

说明：在图形模式下使用中断 33H 的 1 号子功能显示鼠标需要操作系统的配合。如果是 DOS 或 Windows 95/98，屏幕上会出现鼠标箭头标志；如果是 Windows 2000 或 Windows XP，则鼠标箭头显示不出来。

```
Data1   Segment
        NoMouse   db "Mouse isn't Installed!", 13, 10, '$'
        x         db "x:"
        xx        db 3 dup ('0')
        y         db " y:"
        yy        db 3 dup ('0')
        dispmode  db ?
```

```
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;change cx, dx to string and save them in "xx", "yy"
ChangeToStr   proc  uses ax bx
        mov   ax, cx
        mov   bh, 10
        div   bh
        add   ah, '0'
        mov   xx[2], ah
        mov   ah, 0
        div   bh
        add   ah, '0'
        add   al, '0'
        mov   xx[1], ah
        mov   xx, al
        mov   ax, dx
        div   bh
        add   ah, '0'
        mov   yy[2], ah
        mov   ah, 0
        div   bh
        add   ah, '0'
        add   al, '0'
        mov   yy[1], ah
        mov   yy, al
        ret
ChangeToStr   endp
DispPosition  proc  uses ax bx cx dx es bp
        mov   bp, offset x
        push  ds
        pop   es
        mov   ah, 13h
        mov   al, 0
        mov   bh, 0
        mov   bl, 9
        mov   cx, 11
        mov   dx, 0
        int   10h
```

```
        ret
DispPosition  endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ah, 0fh           ;save current display mode
        int   10h
        mov   dispmode, al
        mov   ah, 0             ;set to 640×480×16 mode
        mov   al, 12h
        int   10h
        mov   ax, 0             ;check mouse
        int   33h
        cmp   ax, 0
        jne   Installed
        mov   ah, 9
        mov   dx, offset NoMouse
        int   21h
        jmp   over
Installed:
        mov   ax, 1             ;show mouse
        int   33h
        mov   ah, 1             ;read keyboard status
        int   16h
        jz    NoAnyKey
        cmp   ax, 11bh
        je    over              ;Esc key been pressed
NoAnyKey:
        mov   ax, 3             ;get mouse status
        int   33h
        call  ChangeToStr
        mov   si, cx
        mov   di, dx
        call  DispPosition      ;display mouse position
        jmp   Installed
over:
        mov   ax, 2             ;hide mouse
        int   33h
        mov   ah, 0             ;restore display mode
```

```
        mov   al, dispmode
        int   10h
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

16. 编写一个程序，显示驱动器 C 根目录下所有 EXE 文件。提示：用通配符“*.exe”来查找与此匹配的文件名，找到后，显示之。

**答：**

```
Data1   Segment
        FCB       DB      0, "*       EXE", 26 DUP(?)   ;*与'EXE'之间有7个
                                                          空格
        NewDir    DB '\', 0
        Buffer    DB 128 DUP(?)
        FileName  DB 15 dup(?)
        NoFile    DB "Files not found", 13, 10, '$'
        Err1      DB "No such directory", 13, 10, '$'
        CurDir    DB 256 dup(?)
        CurDrv    DB ?
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
ShowFileName   proc  uses ax cx dx di si es
        push  ds
        pop   es
        cld
        mov   si, offset Buffer[1]
        mov   di, offset FileName
        mov   cx, 8
NextChar:
        movsb
        mov   al, BYTE ptr [si]
        cmp   al, 20h
        je    NameEnd
        loop  NextChar
NameEnd:
        mov   BYTE ptr es:[di], '.'
        inc   di
        mov   si, offset Buffer[9]
```

```
        mov     cx, 3
        rep     movsb
        mov     BYTE ptr es:[di], 13
        mov     BYTE ptr es:[di+1], 10
        mov     BYTE ptr es:[di+2], '$'
        mov     ah, 9
        mov     dx, offset FileName
        int     21h
        ret
ShowFileName    endp
Start:
        mov     ax, Data1
        mov     ds, ax
        mov     ah, 19h                 ;save current dir
        int     21h
        mov     CurDrv, al
        mov     dl, 2                   ;change to driver C
        mov     ah, 0eh
        int     21h
        mov     ah, 47h
        mov     dl, 0
        mov     si, offset CurDir;save current directory
        int     21h
        mov     dx, offset NewDir;change directory to '\'
        mov     ah, 3bh
        int     21h
        jnc     ChangeOK
        mov     dx, offset Err1
        mov     ah, 9
        int     21h
        jmp     Over
ChangeOK:
        mov     dx, offset Buffer;set data transfer area(DTA)
        mov     ah, 1ah
        int     21h
        mov     dx, offset FCB
        mov     ah, 11h
        int     21h
        cmp     al, 0
```

```
        je      Found
        mov     dx, offset NoFile
        mov     ah, 9
        int     21h
        jmp     Over
Found:
        call    ShowFileName
        mov     ah, 12h                 ;find next file
        mov     dx, offset FCB
        int     21h
        cmp     al, 0
        je      Found
Over:
        mov     dl, CurDrv              ;restore current driver
        mov     ah, 0eh
        int     21h
        mov     dx, offset CurDir;restore current directory
        mov     ah, 3bh
        int     21h
        mov     ax, 4c00h
        int     21h
Code1   Ends
    End Start
```

17. 编写一个程序，显示驱动器C根目录下的所有子目录。

**答：**

说明：使用中断21H的11H号子功能搜索子目录要使用扩展FCB。

```
Data1   Segment
; 下面第一个"*"后面7个空格，第二个"*"后面2个空格
        ExpandFCB   DB -1, 5 dup(?), 16, 0, "*       *  ", 26 DUP(0)
        NewDir      DB '\', 0
        Buffer      DB 128 DUP(?)
        FileName    DB 15 dup(?)
        NoFile      DB "Directions not found", 13, 10, '$'
        Err1        DB "No such directory", 13, 10, '$'
        CurDir      db 256 dup(?)
        CurDrv      db ?
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
```

```
GetFileName   proc  uses ax cx dx di si es
        push  ds
        pop   es
        cld
        mov   si, offset Buffer[8]
        mov   di, offset FileName
        mov   cx, 8
NextChar:
        movsb
        mov   al, BYTE ptr [si]
        cmp   al, 20h
        je    NameEnd
        loop  NextChar
NameEnd:
        mov   BYTE ptr es:[di], '.'
        inc   di
        mov   si, offset Buffer[16]
        mov   cx, 3
        rep   movsb
        mov   BYTE ptr es:[di], 0
        mov   BYTE ptr es:[di+1], 13
        mov   BYTE ptr es:[di+2], 10
        mov   BYTE ptr es:[di+3], '$'
        ret
GetFileName   endp
ShowFileName  proc  uses ax dx
        mov   ah, 9
        mov   dx, offset FileName
        int   21h
        ret
ShowFileName  endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ah, 19h                ;save current dir
        int   21h
        mov   CurDrv, al
        mov   dl, 2                  ;change to driver C
        mov   ah, 0eh
```

```
        int   21h
        mov   ah, 47h
        mov   dl, 0
        mov   si, offset CurDir      ;save current directory
        int   21h
        mov   dx, offset NewDir      ;change directory to '\'
        mov   ah, 3bh
        int   21h
        jnc   ChangeOK
        mov   dx, offset Err1
        mov   ah, 9
        int   21h
        jmp   over
ChangeOK:
        mov   dx, offset Buffer      ;set data transfer area(DTA)
        mov   ah, 1ah
        int   21h
        mov   dx, offset ExpandFCB
        mov   ah, 11h
        int   21h
        cmp   al, 0
        je    Found
        mov   dx, offset NoFile
        mov   ah, 9
        int   21h
        jmp   Over
Found:
        call  GetFileName
        mov   ax, 4300h
        mov   dx, offset FileName
        int   21h
        jb    Failure
        and   cx, 10h
        je    Failure
        call  ShowFileName
Failure:
        mov   ah, 12h                ;find next file
        mov   dx, offset ExpandFCB
        int   21h
```

```
        cmp   al, 0
        je    Found
Over:
        mov   dl, CurDrv              ;restore current driver
        mov   ah, 0eh
        int   21h
        mov   dx, offset CurDir       ;restore current directory
        mov   ah, 3bh
        int   21h
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

18. 编写一个程序，设置指定目录下的所有文件为只读文件。

**答：**

```
Data1   Segment
; 下面第一个"*"后面7个空格，第二个后面2个空格
        FCB         DB  0, "*       *  ", 26 DUP(?)
        NewDriver   DB 4                  ;指定的驱动器号为E
        NewDir      DB  '\', 0            ;指定的目录为'\'
        Buffer      DB 128 DUP(?)
        FileName    DB  15 dup(?)
        NoFile      DB  "Files not found", 13, 10, '$'
        Err1        DB "No such directory", 13, 10, '$'
        CurDir      db 256 dup(?)
        CurDrv      db ?
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
GetFileName  proc  uses ax cx dx di si es
        push  ds
        pop   es
        cld
        mov   si, offset Buffer[1]
        mov   di, offset FileName
        mov   cx, 8
NextChar:
        movsb
```

```
        mov     al, BYTE ptr [si]
        cmp     al, 20h
        je      NameEnd
        loop    NextChar
NameEnd:
        mov     BYTE ptr es:[di], '.'
        inc     di
        mov     si, offset Buffer[9]
        mov     cx, 3
        rep     movsb
        mov     BYTE ptr es:[di], 0
        mov     BYTE ptr es:[di+1], 13
        mov     BYTE ptr es:[di+2], 10
        mov     BYTE ptr es:[di+3], '$'
        ret
GetFileName     endp
Start:
        mov     ax, Data1
        mov     ds, ax
        mov     ah, 19h                 ;save current dir
        int     21h
        mov     CurDrv, al
        mov     dl, NewDriver           ;change to driver NewDriver
        mov     ah, 0eh
        int     21h
        mov     ah, 47h
        mov     dl, 0
        mov     si, offset CurDir       ;save current directory
        int     21h
        mov     dx, offset NewDir       ;change directory to NewDir
        mov     ah, 3bh
        int     21h
        jnc     ChangeOK
        mov     dx, offset Err1
        mov     ah, 9
        int     21h
        jmp     over
ChangeOK:
        mov     dx, offset Buffer;set data transfer area(DTA)
```

```
        mov     ah, 1ah
        int     21h
        mov     dx, offset FCB
        mov     ah, 11h
        int     21h
        cmp     al, 0
        je      Found
        mov     dx, offset NoFile
        mov     ah, 9
        int     21h
        jmp     Over
Found:
        call    GetFileName
        mov     ax, 4300h
        mov     dx, offset FileName
        int     21h
        jb      Failure
        or      cx, 1
        mov     ax, 4301h
        mov     dx, offset FileName
        int     21h
Failure:
        mov     ah, 12h                 ;find next file
        mov     dx, offset FCB
        int     21h
        cmp     al, 0
        je      Found
Over:
        mov     dl, CurDrv              ;restore current driver
        mov     ah, 0eh
        int     21h
        mov     dx, offset CurDir;restore current directory
        mov     ah, 3bh
        int     21h

        mov      ax, 4c00h
        int      21h
Code1   Ends
```

```
    End Start
```

19. 编写一个程序，创建一个隐含文件 Data.txt。

答：

```
Data1   Segment
        Buffer      DB 128 DUP(?)
        FileName    DB  "Data.txt", 0
        msg1        DB "Create succeed!", 13,10, '$'
        msg2        DB "Create failure!", 13,10, '$'
        FileHandle  DW ?
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   dx, offset FileName
        mov   cx, 2
        mov   ah, 3ch
        int   21h
        jc    failure
        mov   ax, FileHandle
        mov   dx, offset msg1
        mov   ah, 9
        int   21h
        mov   bx, FileHandle
        mov   ah, 3eh
        int   21h
        jmp   over
failure:
        mov   dx, offset msg2
        mov   ah, 9
        int   21h
over:
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

20. 编写一个程序，把键盘输入写入文件 File.dat 中，类似以下 DOS 命令：

```
COPY    con File.dat
```

答:

```
Data1   Segment
        BufferLen   DB 255
        CharNum     DB  0
        Buffer      DB 256 DUP(?)
        CRLf        db 13, 10, '$'
        FileName    DB  "File.txt", 0
        Hint        DB "Input characters please, press Ctrl-Z to save
                    file, "
                    DB "press Ctrl-C to Cancel.", 13, 10, '$'
        msg1        DB "Create succeed!", 13,10, '$'
        msg2        DB "Create failure!", 13,10, '$'
        FileHandle  DW ?
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;Create a file named by variable "FileName"
;input: none
;output: bl=0ffh, succeed; bl=0, failure; ax=FileHandle if succeed.
CreateFile  proc    uses cx dx
        mov   dx, offset FileName
        mov   cx, 0
        mov   ah, 3ch
        int   21h
        jc    Createfailure
        mov   bl, 0ffh
        jmp   proc_over
Createfailure:
        mov   bl, 0
proc_over:
        ret
CreateFile  endp
;Close a file
;input: bx=File Handle
;output: none
CloseFile   proc    uses ax
        mov     ah, 3eh
        int     21h
```

```
        ret
CloseFile   endp
;copy data in buffer to file, Data length save in CharNum
WriteLine   proc    uses ax bx cx dx
        mov   dx, offset Buffer
        mov   cl, CharNum
        mov   ch, 0
        mov   bx, FileHandle
        mov   ah, 40h
        int   21h
        ret
WriteLine   endp
Start:
        mov   ax, Data1
        mov   ds, ax
        call  CreateFile
        cmp   bl, 0ffh
        jne   failure
        mov   FileHandle, ax
        mov   dx, offset Hint
        mov   ah, 9
        int   21h
next:
        mov   dx, offset BufferLen
        mov   ah, 0ah
        int   21h
        mov   dx, offset CRLF
        mov   ah, 9
        int   21h
        mov   bl, CharNum
        mov   bh, 0
        mov   Buffer[bx+1], 10
        add   bx, 2
        mov   CharNum, bl
        mov   cx, bx
        sub   cx, 2
        mov   bx, 0
next1:
        mov   al, Buffer[BX]
```

```
        cmp   al, 1ah
        je    LastLine
        inc   bx
        loop  next1
        call  WriteLine
        jmp   Next
LastLine:
        mov   CharNum, bl
        call  WriteLine
        mov   bx, FileHandle
        call  CloseFile
        jmp   over
failure:
        mov   dx, offset msg2
        mov   ah, 9
        int   21h
over:
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

21. 使用哪个 DOS 功能可读取中断向量地址？哪个 DOS 功能可把新的中断服务程序地址设置为指定中断的入口地址？

**答：**

使用中断 21H 的 35H 号功能可以读取中断向量的地址，使用中断 21H 的 25H 号功能可以设置中断向量的地址。

22. 编写一个程序，在程序开始运行时，把中断向量表保存到程序的数据段中，在程序结束前把数据段中的备份恢复到系统中去。

**答：**

```
Data1   Segment
        IntTable dw 1024 dup(0)
Data1   Ends
Code1   Segment
        assume  cs:Code1, ds:Data1
SaveIntTable  proc      uses ax bx cx si es
        mov   si, offset IntTable
        mov   cx, 100h
        mov   ah, 35h
```

```
        mov   al, 0
next:
        int   21h
        mov   WORD ptr [si], bx
        mov   WORD ptr [si+2], es
        add   si, 4
        inc   al
        loop  next
        ret
SaveIntTable endp
RestoreIntTable proc  uses ax cx si di es
        mov   ax, 0
        mov   es, ax
        mov   cx, 100h
        mov   si, 0
        mov   di, offset IntTable
next1:
        mov   ax, WORD ptr [si]
        mov   WORD ptr es:[di], ax
        mov   ax, WORD ptr [si+2]
        mov   WORD ptr es:[di+2], ax
        add   si, 4
        add   di, 4
        loop  next1
        ret
RestoreIntTable endp
Start:
        mov   ax, Data1
        mov   ds, ax
        call  SaveIntTable
        call  RestoreIntTable
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

23. 把下列 C 语言的子程序改写成汇编语言子程序。

(1) 取当前光标的位置，并分别存入变量 x 和 y 中。

```
get_cursor(int *x, int *y)
    {union REGS in, out;
```

```
    in.h.ah = 3;                in.h.bh = 0;
    int86(0x10, &in, &out);
    *x = out.h.dh;              *y = out.h.dl;
    }
```

（2）把光标设置在(row, col)位置。

```
locate(int row, int col)
    {union REGS in, out;
    in.h.ah = 2;                in.h.bh = 0;
    in.h.dh = row;              in.h.dl = col;
    int86(0x10, &in, &out);
    }
```

（3）在(row, col)位置显示字符串 string。

```
disp_string(int row, int col, char string[ ])
    {struct  REGPACK  in, out;
        int     x, y;
        get_cursor(&x, &y);
        locate(row, col);
        in.r_ds = FP_SEG(string);    in.r_dx = FP_OFF(string);
        in.r_ax = 0x900;
        intr(0x21, &in);
        locate(x, y);
    }
```

**答：**

（1）

```
;返回值存放在 dh 和 dl 中，dh 对应*x，DL 对应*y。
get_cursor  proc    uses ax bx
        mov   ah, 3
        mov   bh, 0
        int   10h
        ret
get_cursor  endp
```

（2）

```
;参数 row 通过寄存器 dh 传递，参数 col 通过寄存器 dl 传递中。
locate  proc    uses ax bx cx dx
        mov     ah, 2
```

```
            mov     bh, 0
            int     10h
            ret
    locate   endp
```

(3)

```
    disp_string   proc
            call  get_cursor
            push  dx
            mov   dx, col
            mov   cx, row
            mov   dh, cl
            call  locate
            mov   ax, seg string
            mov   ds, ax
            mov   dx, offset string
            mov   ax, 900h
            int   21h
            pop   dx
            call  locate
            ret
    disp_string   endp
```

# 第九章　宏

1．在宏定义时，使用的关键字是什么？宏名是否需要成对出现？

**答：**

宏定义时使用的关键字是 MACRO 和 ENDM。宏名不需要成对出现。

2．在宏引用时，是否要求实参与形参的个数相等？若不要求，请简述当二者个数不一致时，会出现什么情况。

**答：**

在宏引用中，实参的位置要与形参的位置要对应，但实参的个数可以与形参的个数不相同。当实参的个数多于形参的个数时，多出的实参被忽略；当实参的个数少于形参的个数时，没有实参对应的形参用“空”来对应。但在宏展开时，所得到的指令必须是合法的汇编指令；否则，汇编程序将会给出错误信息。

3．宏和子程序的主要区别有哪些？一般在什么情况下选用宏较好，在什么情况下选用子程序较好？

**答：**

宏和子程序都是为了简化源程序的编写，提高程序的可维护性，但是它们二者之间存在着以下本质的区别：

（1）在源程序中，通过书写宏名来引用宏，而子程序是通过 CALL 指令来调用；

（2）汇编程序对宏通过宏扩展来加入其定义体，宏引用多少次，就相应扩展多少次，所以，引用宏不会缩短目标程序；而子程序代码在目标程序中只出现一次，调用子程序是执行同一程序段，因此，目标程序也得到相应的简化；

（3）宏引用时，参数是通过“实参”替换“形参”来实现传递的，参数形式灵活多样，而子程序调用时，参数是通过寄存器、堆栈或约定存储单元进行传递的；

（4）宏引用语句扩展后，目标程序中就不再有宏引用语句。运行时，不会有额外的时间开销，而子程序的调用指令存在于目标程序之中，执行调用指令需要时间。

总之，当程序片段不长，速度是关键因素时，可采用宏来简化源程序，但当程序片段较长，存储空间是关键因素时，可采用子程序的方法来简化源程序和目标程序。

4．宏的参数是如何传入宏定义体的？宏的参数传递与子程序的参数传递有哪些区别？

**答：**

在引用宏时，参数是通过“实参”替换“形参”的方式来实现传递的，参数的形式灵

活多样，而子程序调用时，参数是通过寄存器、堆栈或约定存储单元进行传递的。

5. 在有标号的宏定义体中，为什么最好使用 LOCAL 伪指令来说明标号？它在宏定义体中应处于什么位置？

**答：**

在宏定义体中，如果存在标号，则该标号要用伪指令 LOCAL 说明为局部标号，否则，当在源程序中，有多于一次引用该宏时，汇编程序在进行宏扩展后将会给出：标号重复定义的错误。

伪指令 LOCAL 必须是伪指令 MACRO 后的第一条语句，在 MACRO 和 LOCAL 之间也不允许有注释和分号标志。

6. 子程序和宏中的 LOCAL 伪指令的作用有哪些不同？

**答：**

在子程序中，LOCAL 用来说明局部变量；在宏定义中，LOCAL 用来说明局部标号。

在子程序的不同调用中，用 LOCAL 定义的局部变量名保持不变；而在每次宏引用的扩展中，将会自动产生出一个惟一的局部标号。

7. 编写 32 位相加的宏 ADD32，它把 32 位寄存器组 BX-AX 加到 DX-CX 中。

**答：**

```
add32   macro
        add dx, bx
        adc cx, ax
        endm
```

8. 编写符号扩展的宏 CBD，它将存于 AL 中的有符号数扩展成 ECX-EBX 中 64 位有符号数（其中：ECX 是 64 位有符号数的高位）。

**答：**

```
cbd     macro
        cbw
        cwde
        cdq
        mov     ecx, edx
        mov     ebx, eax
        endm
```

9. 编写英文字母大小写互换的宏 Exchange，其有一个形参，允许字节型的寄存器或存储单元作为实参来引用。

**答：**

```
  exchange macro    oprd
ii        mov     al,oprd
          .if  (al>='A')&&(al<='Z')
```

```
            add   oprd,20h
        .elseif  (al>='a')&&(al<='z')
            sub oprd, 20h
        .endif
        endm
```

10. 编写一个宏“AddList Para1, Para2, num”，其功能是将从 Para2 开始的内存单元的值加到以 Para1 开始的内存单元中，num 是相加的字节数。

答：

```
AddList macro   Para1, Para2, num
        Local  next
        mov    cx, num
        mov    si, Para1
        mov    di, Para2
next:
        mov    al, BYTE ptr [si]
        add    al, BYTE ptr [di]
        inc    si
        inc    di
        loop   next
        endm
```

11. 编写一个宏“SUM Data, Length, Result”，其功能是求从 Data 开始的无符号字节累加和，并把结果存入字类型参数 Result 中，Length 是需要累加的字节数。

答：

```
Sum  macro  Data, Length, Result
     local  next
     mov    cx, Length
     mov    bx, Data
     mov    ax, 0
     mov    dx, 0
next:
    mov  dl, BYTE ptr [bx]
    add  ax, dx
    inc  bx
    loop next
    mov  Result, ax
    endm
```

12. 编写一个宏来定义 26 个大写字母表。

答：

```
AlphaTable   macro   Table
        Table   label byte
        Count='A'
        rept 26
             db  Count
             Count=Count+1
        endm
```

13. 编写一个宏，它产生 n 条 NOP 指令，其中 n 是宏的形式参数。

答：

```
REPT  n
      NOP
      ENDM
```

14. INCLUDE 指示符的作用是什么?

答：

MASM 6.11 系统定义了大量的标准宏，程序员能很方便地使用它们。在使用这些系统宏之前，要像 C 语言那样用伪指令 INCLUDE 把有关“宏库”文件包含在用户的源程序中，程序员也可以用 INCLUDE 包括自己定义的宏库。

15. 编写只有一个形式参数的宏 PRINT，其具体功能如下：

1）若引用时带有参数，则在屏幕上显示其参数字符，如：PRINT A，则显示字符 A；

2）若引用时不带实参，则显示回车和换行，如：PRINT。

提示：用 IFB 或 IFNB 语句来测试是否有参数。

答：

```
PRINT   macro  para1
        IFB    <para1>
            mov  ah, 2
            mov  dl, 13
            int  21h
            mov  ah, 2
            mov  dl, 10
            int  21h
        else
            mov  ah, 2
            mov  dl, para1
            int  21h
        endif
        endm
```

# 第十章　应用程序的设计

1．编写一个子程序，实现字符串的逆转。如：ABCD→DCBA。

**答：**

```
;子程序功能：逆转字符串
;Input: bx—address of string, cx—string length
;Output: none
Reverse proc  uses ax cx si di
        mov   si, bx
        mov   di, bx
        add   di, cx
        dec   di
        shr   cx, 1
next:
        mov   al, BYTE ptr [si]
        mov   ah, BYTE ptr [di]
        mov   BYTE ptr [di], al
        mov   BYTE ptr [si], ah
        inc   si
        dec   di
        loop  next
        ret
Reverse endp
```

2．编写一段程序，接收 4 位十六进制数，然后用移位的方法把它转换成八进制数并输出在屏幕上。

**答：**

```
Data1   Segment
  buffer  db  6 dup(?)
  crlf    db  13, 10, '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;子程序功能：将 4 位十六进制数转换成八进制数形式的字符串保存在 buffer 中
;Input: ax—number
;Output: none
```

```
HtoQ    proc  uses bx cx dx
        mov   cx, 6
        mov   bx, 0
next1:
        mov   buffer[bx], 20h
        inc   bx
        loop  next1
        mov   bx, 5
next2:
        mov   dx, 7
        and   dx, ax
        add   dx, 30h
        mov   buffer[bx], dl
        dec   bx
        shr   ax, 3  ;8086以后的CPU指令，移位位数直接写在指令中
        cmp   ax, 0
        je    over
        jmp   next2
over:
        ret
HtoQ    endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ax, 0ffffh
        call  HtoQ
        mov   ah, 9
        mov   dx, offset buffer
        int   21h
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

3. 编写一段程序，以十六进制的形式显示内存 0400H:1000H 开始的 100 字，要求每行显示 16 个字，每个字之间有空格。

**答：**

```
Data1   Segment
    buffer  db 2 dup(?), 20h, '$'
    crlf    db 13, 10, '$'
```

```
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;子程序功能：将一个字节的数据转换成两位十六进制字符串，保存在buffer中
;Input: al--number
;Output: none
HexToStr        proc  uses ax cx
        push    ax
        push    cx
        mov     ah, al
        mov     cx, 4
        shr     ah, cl
        and     al, 15
        cmp     ah, 9
        jbe     AH_below_9
        add     ah, 55
        mov     buffer, ah
        jmp     next
AH_below_9:
        add     ah, 30h
        mov     buffer, ah
next:
        cmp     al, 9
        jbe     AL_below_9
        add     al, 55
        mov     buffer[1], al
        jmp     over
AL_below_9:
        add     al, 30h
        mov     buffer[1], al
over:
        pop     cx
        pop     ax
        ret
HexToStr        endp
Start:
        mov     ax, Data1
        mov     ds, ax
        mov     ax, 0400h
```

```
        mov     es, ax
        mov     bx, 1000h
        mov     si, 0
NextChar:
        mov     al, BYTE ptr es:[bx+si]
        call    HexToStr
        mov     ah, 9
        mov     dx, offset buffer
        int     21h
        inc     si
        mov     dx, 15
        and     dx, si
        jnz     NoLineFeed
        mov     ah, 9
        mov     dx, offset crlf
        int     21h
NoLineFeed:
        cmp     si, 100
        jne     NextChar
        mov     ax, 4c00h
        int     21h
Code1   Ends
        End Start
```

4．简述 COM 文件的主要特点，它与 EXE 文件有哪些差异？

**答：**

COM 文件与 EXE 文件的主要差异在于：

（1）COM 文件只有一个段，其字节数不会超过 64K；而 EXE 文件可以有多个段。

（2）当操作系统装入 COM 文件时，4 个段寄存器（CS、DS、ES 和 SS）都用 PSP 的段值来初始化，它们的值相同；而 EXE 文件并不要求段寄存器的值相同。

（3）COM 文件必须用伪指令 ORG 100H 来说明空出前 256 个字节；而 EXE 文件不需要。

5．编写一个显示“Hello”的程序，试比较它的 COM 格式文件和 EXE 格式文件的大小。

**答：**

EXE 文件对应的源文件如下：

```
Data1   Segment
        szMsg db 'Hello World!',13,10,'$'
```

```
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
Start:
        mov   ax, Data1
        mov   ds, ax
        lea   dx,szMsg
        mov   ah,9
        int   21h
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

EXE 文件的大小为 546 个字节。

COM 文件对应的源文件如下：

```
Code1   Segment
        org   100H
Start:
        lea   dx,szMsg
        mov   ah,9
        int   21h
        mov   ax, 4c00h
        int   21h
        szMsg db 'Hello World!',13,10,'$'
Code1   Ends
        End Start
```

COM 文件的大小为 28 个字节。

6. 编写一个在屏幕上连续显示“Welcome...”的信息，要求显示颜色和显示位置都是随机的。在新位置显示时，原位置的信息抹去。当按任意键时，程序结束运行。

**答：**

```
Data1   Segment
    Welcome     db "Welcome..."
    CurrentPosi dw 0
    blank       db "          "
    tempnum     db  ?
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
```

```
;在屏幕上显示一个字符串
;Input: dh—row number, dl—column number,bl—color, bp—string address
;Output: none
ShowMsg proc  uses ax bx cx es
        push   es
        push   cx
        push   bx
        push   ax
        mov    ax, ds
        mov    es, ax
        mov    cx, 10
        mov    bh, 0
        mov    ah, 13h
        mov    al, 0
        int    10h
        ret
ShowMsg endp
;擦除屏幕上的 Welcome...
;Input: CurrentPosi(上次显示 Welcome...时的位置)
;Output: none
EraseMsg proc uses bx dx bp
        mov   dx, CurrentPosi
        mov   bl, 0
        mov   bp, offset blank
        call  ShowMsg
        ret
EraseMsg    endp
;显示 Welcome...
;Input: dh——row number, dl—column number, bl—color
;Output: CurrentPosi(输入的 AX 的值)
ShowWelcome proc  uses bp
        Call   EraseMsg
        mov    CurrentPosi, dx
        mov    bp, offset Welcome
        mov    cx, 10
        call   ShowMsg
        ret
ShowWelcome endp
```

```
;生成一个伪随机数的行号(0~24之内)、列号(0~80之内)和颜色(1~15)
;Input: none
;Output: dh--row number, dl--column number, bl--color
Random      proc
        mov   al, tempnum
        add   al, 24
        mov   ah, 13
        mul   ah
        mov   tempnum, al
        mov   bh, 25
        div   bh
        mov   dh, ah          ;row number
        mov   al, tempnum
        add   al, 22
        mov   ah, 7
        mul   ah
        mov   tempnum, al
        mov   bh, 80
        div   bh
        mov   dl, ah          ;column number
        mov   al, tempnum
        add   al, 8
        mov   ah, 3
        mul   ah
        mov   tempnum, al
        mov   bh, 14
        div   bh
        inc   ah
        mov   bl, ah          ;color
        ret
Random      endp
Start:
        mov   ax, Data1
        mov   ds, ax
next:
        mov   ah, 1           ;read keyboard status
        int   16h
        jnz   KeyPressed
        call  Random
```

```
        call  ShowWelcome
        jmp   next
KeyPressed:
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

7. 从键盘输入一个表示年份的正整数（1~65 535），然后判断其是否为闰年。若是，则输出“Yes”，否则，输出“No”。

**答：**

说明：简化起见，子程序 GetYear 没有对输入时可能出现的空格进行识别和排除。该子程序的功能是分析输入的字符串是否全由“0”~“9”类型的字符构成，并判断数字是否超出 65 535，如果超出当作无效。

```
Data1   Segment
        BufferLen  DB 6
        CharNum DB 0
        Buffer     DB 6 DUP(?)
        Year       dw 0
        yes        db 13, 10, "Yes!", 13, 10, '$'
        no         db 13, 10, "No!", 13, 10, '$'
        void       db 13, 10, "Invalid data!", 13, 10, '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;Change string to a number between 1~65 535. if invalid, return 0
;input: CharNum-String Length, Buffer-string
;Output: ax—the number
GetYear     proc  uses bx cx dx si
        mov   cl, CharNum
        mov   ch, 0
        mov   ax, 0
        mov   si, 0
        mov   dx, 0
next:
        cmp   cx, 0
        je    over
        mov   bx, 10
        mul   bx
```

```
        cmp   dx, 0
        jne   invalid
        mov   bl, Buffer[si]
        cmp   bl, '0'
        jb    invalid
        cmp   bl, '9'
        ja    invalid
        sub   bl, '0'
        mov   bh, 0
        add   ax, bx
        inc   si
        dec   cx
        jmp   next
invalid:
        mov   ax, 0
over:
        ret
GetYear     endp

Start:
    mov     ax, Data1
        mov   ds, ax
        mov   dx, offset BufferLen
        mov   ah, 0ah
        int   21h
        call  GetYear
        cmp   ax, 0
        je    _invalid
        mov   Year, ax
        mov   dx, 0
        mov   bx, 400
        div   bx
        cmp   dx, 0
        je    _yes
        mov   dx, 0
        mov   ax, Year
        mov   bx, 100
        div   bx
        cmp   dx, 0
```

```
        je    _no
        mov   dx, 0
        mov   ax, Year
        mov   bx, 4
        div   bx
        cmp   dx, 0
        je    _yes
_no:
        mov   dx, offset No
        jmp   showmsg
_invalid:
        mov   dx, offset void
        jmp   showmsg
_yes:
        mov   dx, offset yes
showMsg:
        mov   ah, 9
        int   21h
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

8. 编写一个带命令行参数的程序 Words，输出指定正文文件中的单词，假设单词为连续的字母串。

比如：words file.txt，显示文件 file.txt 中的每个单词

**答：**

说明：下面的程序可以根据文件中的制表符、空格、回车、换行符识别单词。

```
Data1   Segment
    ok      db "OK!", 13, 10, '$'
    fail    db "File cannot be opened!", 13, 10, '$'
    handle  dw ?
    GetWord db 0          ;用于判断当前是否找到一个单词，是则为 1，反之为 0
    buffer  db  2
    errmsg1 db "Read file error!", 13, 10, '$'
    _crlf   db 13, 10, '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;显示回车换行字符
```

```
crlf    proc  uses ax dx
        mov   ah, 9
        mov   dx, offset _crlf
        int   21h
        ret
crlf        endp
Start:
        mov   ah, 51h               ;get psp
        int   21h
        mov   ds, bx
        mov   si, 80h
        mov   bl, [si]
        add   bl, 81h
        mov   bh, 0
        mov   BYTE ptr [bx], 0  ;prepare for open file
        mov   ah, 3dh
        mov   al, 0
        mov   dx, 82h
        int   21h                   ;open file in read mode
        jc    error
        mov   bx, Data1
        mov   ds, bx
        mov   handle, ax
        mov   GetWord, 0
next:
        mov   ah, 3fh               ;read a char from file
        mov   bx, handle
        mov   cx, 1
        mov   dx, offset buffer
        int   21h
        jc    readerr
        cmp   ax, 0
        je    EndOfFile
        mov   dl, [buffer]
        cmp   dl, ' '
        je    EndPoint
        cmp   dl, 9
        je    EndPoint
        cmp   dl, 13
```

```
        je    EndPoint
        cmp   dl, 10
        je    EndPoint
        mov   ah, 2              ;display a char
        int   21h
        mov   GetWord, 1
        jmp   next
EndPoint:
        mov   al, GetWord
        cmp   al, 0
        je    next
        call  crlf
        mov   GetWord, 0
        jmp   next
readerr:
        mov   ah, 9              ;显示读文件出错信息
        mov   dx, offset errmsg1
        int   21h
        jmp   closefile
EndOfFile:
        call  crlf
closefile:
        mov   ah, 3eh                ;close file
        mov   bx, handle
        int   21h
        jmp   finish
error:
        mov   ax, Data1
        mov   ds, ax
        mov   dx, offset fail
        mov   ah, 9                  ;显示打开文件出错信息
        int   21h
finish:
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

9. 编写一个建立双向链表的程序，每接受一个整数，链表增加一个结点，当遇到负数时，结束链表结点的增加，然后从表尾向前输出各结点中的数值。

答：

```
Data1   Segment
    Alloc_Err_Msg   db "Alloc memory block failure!", 13, 10, '$'
    Modify_Err_Msg  db "Modify memory block failure!", 13, 10, '$'
    Remove_Err_Msg  db "Remove memory block failure!", 13, 10, '$'
    Head            dw 0              ;链表的头指针，初值为 0
    CurPoint        dw 0              ;链表的尾指针
    PrePoint        dw 0              ;链表尾节点的后向指针
    DataArray       dw 100, 200, 300, 400, -1
    buffer          db 5 dup(?), ' ', '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;将字数据转换成十进制无符号形式的字符串，存放在 buffer 中
;Input: AX—Data
;Output: buffer—转化后的十进制无符号形式的字符串
IntToStr proc uses ax bx cx dx
        mov   bx, 0
        mov   cx, 5
Next1:
        mov   buffer[bx], ' '       ;清理 buffer
        inc   bx
        loop   Next1
        mov   cx, 10
        mov   bx, 4
Next2:
        mov   dx, 0
        div   cx
        add   dl, 30h
        mov   buffer[bx], dl
        cmp   ax, 0
        je    IntToStr_Return
        dec   bx
        jmp   Next2
IntToStr_Return:
        ret
IntToStr endp
```

```
;在链表中增加一个新节点，申请1个字节的内存，用于存放数据、前向指针和后向指针
;节点数据格式：Data(DW)+ForwardPoint(DW)+BackwardPoint(DW)
;Input: DX—Data, Head—链表的头指针,CurPoint—链表的尾指针
;Output: Head—当生成链表的第一个节点时返回头指针
;                  CurPoint—链表的尾指针
;                  AL—内存申请是否成功，-1表示失败，0表示成功
AddPoint      proc  uses bx ex
        mov   ah, 48h            ;申请1个字节的内存
        mov   bx, 1
        int   21h
        jc    AllocErr
        mov   bx, CurPoint
        mov   PrePoint, bx
        mov   CurPoint, ax
        cmp   Head, 0
        jne   WriteData
        mov   Head, ax
        mov   PrePoint, ax
WriteData:
        mov   es, ax          ;填写新节点的数据
        mov   bx, 0
        mov   WORD ptr es:[bx], dx
        mov   ax, Head
        mov   WORD ptr es:[bx+2], ax
        mov   ax, PrePoint
        mov   WORD ptr es:[bx+4], ax
        ;判断是否需要修改增加节点前链表最后一个节点的前向指针
        cmp   ax, CurPoint
        je    NoNeed
        mov   es, ax
        mov   ax, CurPoint
        mov   WORD ptr es:[bx+2], ax
NoNeed:
        mov   al, 0
        jmp   AddPoint_Return
AllocErr:
        mov   al, -1
```

```
        jmp   AddPoint_Return
AddPoint_Return:
        ret
AddPoint      endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov     bx, 0
NextData:
        mov   dx, DataArray[bx]
        cmp   dx, 0
        jl    DispDataArray
        call  AddPoint
        cmp   al, 0
        je    AddSucceed
        mov   ah, 9                          ;失败则停止增加链表节点
        mov   dx, offset Alloc_Err_Msg
        int   21h
        jmp   DispDataArray
AddSucceed:
        add   bx, 2
        jmp   NextData
DispDataArray:
        cmp   Head, 0
        je    Finish                         ;如果是空链表则直接结束
        mov   ax, Head
NextPoint:
        mov   es, ax
        mov   bx, 0
        mov   ax, WORD ptr es:[bx]           ;取节点数据并显示
        call  IntToStr
        mov   ah, 9
        mov   dx, offset buffer
        int   21h
        mov   ax, WORD ptr es:[bx+2]         ;取下一个节点的地址
        cmp   ax, Head
        jne   NextPoint
```

```
ReleaseMemory:
        mov   dx, CurPoint
RemoveNextPoint:
        mov   es, dx
        mov   bx, 4
        mov   cx, WORD ptr es:[bx]        ;读前一个节点的地址
        mov   ah, 49h
        int   21h
        jc    RemoveErr
        cmp   dx, Head
        je    Finish
        mov   dx, cx
        jmp   RemoveNextPoint
RemoveErr:
        mov   ah, 9
        mov   dx, offset Remove_Err_Msg
        int   21h
        jmp   Finish
Finish:
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

10. 编写一个带命令行参数的程序 Calc，其命令行参数是一些整数，程序输出它们之和。如果某参数为非法整数，则忽略之。

比如：Calc 120 +34 –56，则程序显示 98。

答：

```
Data1   Segment
    result        dw 0
    temp          dw 0
    symbol        dw 0;符号位，0 为正，-1 为负
    buffer        db 5 dup(?), 13, 10, '$'
    InvalidMsg    db "Invalid parameters!", 13, 10, '$'
    OverflowMsg   db "Result overflow!", 13, 10, '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;根据 symbol 的值计算 rusult 和 temp 的和或差，结果存在 result 中
```

```
;同时将 symbol 和 temp 置 0
Calculate   proc  uses ax
        push  ax
        mov   ax, result
        cmp   symbol, 0
        je    _add
        sub   ax, temp
        jmp   CalculateReturn
_add:
        add   ax, temp
CalculateReturn:
        mov   result, ax
        mov   symbol, 0
        mov   temp, 0
        pop   ax
        ret
Calculate   endp
;将字数据转换成十进制无符号形式的字符串，存放在 buffer 中
;Input: AX—Data
;Output: buffer—转化后的十进制无符号形式的字符串
IntToStr    proc  uses ax bx cx dx
        mov   bx, 0
        mov   cx, 5
Next1:
        mov   buffer[bx], ' '              ;清理 buffer
        inc   bx
        loop  Next1
        mov   cx, 10
        mov   bx, 4
Next2:
        mov   dx, 0
        div   cx
        add   dl, 30h
        mov   buffer[bx], dl
        cmp   ax, 0
        je    IntToStr_Return
        dec   bx
        jmp   Next2
```

```
IntToStr_Return:
        ret
IntToStr      endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ah, 51h           ;get psp
        int   21h
        mov   es, bx
        mov   si, 81h
NextChar:
        inc   si
        mov   al, BYTE ptr es:[si]
        cmp   al, 13
        jne   NotCR
        call  Calculate
        jo    Overflow
        jmp   OutputResult
NotCR:
        cmp   al, ' '
        jne   NotBlank
        call  Calculate
        jo    Overflow
        jmp   NextChar
NotBlank:
        cmp   al, '+'
        jne   NotAdd
        jmp   NextChar
NotAdd:
        cmp   al, '-'
        jne   NotSub
        xor   symbol, 0ffffh
        jmp   NextChar
NotSub:
        cmp   al, '0'
        jb    InvalidChar
        cmp   al, '9'
        ja    InvalidChar
        mov   cl, al
```

```
        sub   cl, '0'
        mov   ch, 0
        mov   ax, temp
        mov   bx, 10
        mul   bx
        add   ax, cx
        adc   dx, 0
        cmp   dx, 0
        jne   Overflow
        mov   temp, ax
        jmp   Nextchar
InvalidChar:
        mov   ah, 9
        mov   dx, offset InvalidMsg
        int   21h
        jmp   finish
Overflow:
        mov   ah, 9
        mov   dx, offset OverflowMsg
        int   21h
        jmp   finish
OutputResult:
        mov   ax, result
        call  IntToStr
        mov   ah, 9
        mov   dx, offset buffer
        int   21h
finish:
        mov   ax, 4c00h
        int   21h
Code1   Ends
        End Start
```

11. 在第 10 题的基础上，不考虑运算符的优先级，实现一个可进行算术四则运算（只有加、减、乘和除，没有括号等）的计数器，并显示其计算结果。

**答：**

```
Data1   Segment
    result      dw  0
    temp        dw 0
    symbol      db '+'  ;运算符号位
```

```
    buffer      db 5 dup(?), 13, 10, '$'
    InvalidMsg  db "Invalid parameters!", 13, 10, '$'
    OverflowMsg db "Result overflow!", 13, 10, '$'
    divZeroMsg  db "Divide Zero Error!", 13, 10, '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;根据 symbol 的值计算 rusult 和 temp 的运算结果，存放在 result 中，同时将 temp 置 0
;如果除法运算中除数为 0，则 CX 设置为-1；反之为 0。
Calculate   proc  uses ax bx dx
        mov   cx, 0
        mov   ax, result
        mov   bh, symbol
        cmp   bh, '+'
        je    _add
        cmp   bh, '-'
        je    _sub
        cmp   bh, '*'
        je    _mul
        cmp   bh, '/'
        jne   CalculateReturn
        cmp   temp, 0
        jne   _div
        mov   cx, -1
        jmp   CalculateReturn
_div:
        mov   dx, 0
        idiv  temp
        jmp   CalculateReturn
_add:
        add   ax, temp
        jmp   CalculateReturn
_sub:
        sub   ax, temp
        jmp   CalculateReturn
_mul:
        imul  temp
CalculateReturn:
        mov   result, ax
```

```
        mov   temp, 0
        ret
Calculate   endp
;将字数据转换成十进制无符号形式的字符串，存放在 buffer 中
;Input: AX—Data
;Output:buffer—转化后的十进制无符号形式的字符串
IntToStr    proc  uses ax bx cx dx
        mov   bx, 0
        mov   cx, 5
Next1:
        mov   buffer[bx], ' '        ;清理 buffer
        inc   bx
        loop  Next1
        mov   cx, 10
        mov   bx, 4
Next2:
        mov   dx, 0
        div   cx
        add   dl, 30h
        mov   buffer[bx], dl
        cmp   ax, 0
        je    IntToStr_Return
        dec   bx
        jmp   Next2
IntToStr_Return:
        ret
IntToStr    endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ah, 51h            ;get psp
        int   21h
        mov   es, bx
        mov   si, 81h
NextChar:
        inc   si
        mov   al, BYTE ptr es:[si]
        cmp   al, 13
        jne   NotCR
```

```
        call  Calculate
        jo    Overflow
        cmp   cx, -1
        je    DivZero
        jmp   OutputResult
NotCR:
        cmp   al, ' '
        jne   NotBlank
        jmp   NextChar
NotBlank:
        cmp   al, '+'
        je    CalcChar
        cmp   al, '-'
        je    CalcChar
        cmp   al, '*'
        je    CalcChar
        cmp   al, '/'
        jne   NotCalcChar
CalcChar:
        call  Calculate
        jo    Overflow
        cmp   cx, -1
        je    DivZero
        mov   symbol, al
        jmp   NextChar
NotCalcChar:
        cmp   al, '0'
        jb    InvalidChar
        cmp   al, '9'
        ja    InvalidChar
        mov   cl, al
        sub   cl, '0'
        mov   ch, 0
        mov   ax, temp
        mov   bx, 10
        mul   bx
        add   ax, cx
        adc   dx, 0
        cmp   dx, 0
```

```
        jne   Overflow
        mov   temp, ax
        jmp   Nextchar
InvalidChar:
        mov   ah, 9
        mov   dx, offset InvalidMsg
        int   21h
        jmp   finish
Overflow:
        mov   ah, 9
        mov   dx, offset OverflowMsg
        int   21h
        jmp   finish
DivZero:
        mov   ah, 9
        mov   dx, offset divZeroMsg
        int   21h
        jmp   finish
OutputResult:
        mov   ax, result
        call  IntToStr
        mov   ah, 9
        mov   dx, offset buffer
        int   21h
finish:
        mov ax, 4c00h
        int 21h
Code1   Ends
    End Start
```

12. 编写一个简单的加密程序，它把一个文件中的所有字母字符按下列规律进行转换，非字母的字符保持不变。

转换规律：'A'→'Z'，'B'→'Y'，……。

**答：**

```
Data1   Segment
    err1        db "File cannot be opened!", 13, 10, '$'
    err2        db "Temporary file cannot be opened!", 13, 10, '$'
    err3        db "Read file error!", 13, 10, '$'
    err4        db "delete source file failure", 13, 10, '$'
    err5        db "Change file name failure!", 13, 10, '$'
```

```
    handle      dw ?
    TmpHandle   dw  ?       ;temporary file handle
    buffer      db  2
    FileName    DB  "~encrypt.tmp", 0
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
ShowMsg macro  MsgName
        mov   ax, Data1
        mov   ds, ax
        mov   dx, offset MsgName
        mov   ah, 9
        int   21h
endm
;Create a temporary file for encrypt, filename saved in "FileName"
;input: none
;output:bl=0ffh, succeed; bl=0, failure.
CreateTempFile  proc  uses ax cx dx
        mov   dx, offset FileName
        mov   cx, 0
        mov   ah, 3ch
        int   21h
        jc    Createfailure
        mov   bl, 0ffh
        mov   TmpHandle, ax
        jmp   proc_over
Createfailure:
        mov   bl, 0
proc_over:
        ret
CreateTempFile  endp
Start:
        mov   ah, 51h           ;get psp
        int   21h
        mov   ds, bx
        mov   si, 80h
        mov   bl, [si]
        add   bl, 81h
        mov   bh, 0
```

```
        mov   BYTE ptr [bx], 0  ;prepare for open file
        mov   ah, 3dh
        mov   al, 0
        mov   dx, 82h
        int   21h                       ;open file in read mode
        jc    error1
        mov   bx, Data1
        mov   ds, bx
        mov   handle, ax
        call  CreateTempFile
        cmp   bl, 0ffh
        jne   error2
next:
        mov   ah, 3fh                   ;read a char from file
        mov   bx, handle
        mov   cx, 1
        mov   dx, offset buffer
        int   21h
        jc    error3
        cmp   ax, 0
        je    closefile
        mov   dl, [buffer]
        .if(dl>='A')&&(dl<='Z')
        sub   dl, 'a'
        mov   dh, 25
        sub   dh, dl
        mov   dl, 'a'
        add   dl, dh
        .elseif(dl>='a')&&(dl<='z')
        sub   dl, 'A'
        mov   dh, 25
        sub   dh, dl
        mov   dl, 'A'
        add   dl, dh
        .endif
        mov   [buffer], dl
        mov   dx, offset buffer
        mov   cx, 1
        mov   bx, TmpHandle
```

```
        mov   ah, 40h
        int   21h
        jmp   next
closefile:
        mov   ah, 3eh                ;close file
        mov   bx, handle
        int   21h
        call  CloseTempFile
        mov   ah, 51h                ;get psp
        int   21h
        mov   ds, bx
        mov   ah, 41h
        mov   dx, 82h
        int   21h                    ;open file in read mode
        jc    error4
        push  ds
        pop   es
        mov   di, 82h
        mov   ax, Data1
        mov   ds, ax
        mov   dx, offset FileName
        mov   ah, 56h
        int   21h
        jc    error5
        jmp   finish
error1:
        ShowMsg err1
        jmp     finish
error2:
        ShowMsg err2
        jmp     finish
error3:
        ShowMsg err3
        jmp     finish
error4:
        ShowMsg   err4
        jmp       finish
error5:
        ShowMsg   err5
```

```
finish:
        mov   ax, 4c00h
        int   21h
Code1   Ends
    End Start
```

13. 编写一个程序，它可显示 BMP 的图形文件。

**答：**

编写 BMP 文件的显示程序比较复杂，在此只给出 BMP 文件的文件格式，供参考。

（1）BMP 文件组成

BMP 文件由文件头、位图信息头、颜色信息和图形数据 4 部分组成。

（2）BMP 文件头

BMP 文件头数据结构含有 BMP 文件的类型、文件大小和位图起始位置等信息。

其结构定义如下：

```
typedef struct tagBITMAPFILEHEADER{
    WORDbfType;              // 位图文件的类型，必须为 BM
    DWORD bfSize;            // 位图文件的大小，以字节为单位
    WORDbfReserved1;         // 位图文件保留字，必须为 0
    WORDbfReserved2;         // 位图文件保留字，必须为 0
    DWORD bfOffBits;         // 位图数据的起始位置，以相对于位图
                             // 文件头的偏移量表示，以字节为单位
} BITMAPFILEHEADER;
```

（3）位图信息头

BMP 位图信息头数据用于说明位图的尺寸等信息。

```
typedef struct tagBITMAPINFOHEADER{
    DWORD biSize;            // 本结构所占用字节数
    LONGbiWidth;             // 位图的宽度，以像素为单位
    LONGbiHeight;            // 位图的高度，以像素为单位
    WORD biPlanes;           // 目标设备的级别，必须为 1
    WORD biBitCount          // 每个像素所需的位数，必须是 1（双色）
                             //4（16 色），8（256 色）或 24（真彩色）之一
    DWORD biCompression;     // 位图压缩类型，必须是 0（不压缩）
                             //1（BI_RLE8 压缩类型）或 2（BI_RLE4 压缩类型）
                               之一
    DWORD biSizeImage;       // 位图的大小，以字节为单位
```

```
    LONGbiXPelsPerMeter;        // 位图水平分辨率，每米像素数
    LONGbiYPelsPerMeter;        // 位图垂直分辨率，每米像素数
    DWORD biClrUsed;            // 位图实际使用的颜色表中的颜色数
    DWORD biClrImportant;       // 位图显示过程中重要的颜色数
} BITMAPINFOHEADER;
```

（4）颜色表

颜色表用于说明位图中的颜色，它有若干个表项，每一个表项是一个 RGBQUAD 类型的结构，定义一种颜色。RGBQUAD 结构的定义如下：

```
typedef struct tagRGBQUAD {
    BYTErgbBlue;                // 蓝色的亮度（值范围为 0~255）
    BYTErgbGreen;               // 绿色的亮度（值范围为 0~255）
    BYTErgbRed;                 // 红色的亮度（值范围为 0~255）
    BYTErgbReserved;            // 保留，必须为 0
} RGBQUAD;
```

颜色表中 RGBQUAD 结构数据的个数由 biBitCount 来确定：

当 biBitCount 为 1、4、8 时，分别有 2、16、256 个表项；

当 biBitCount=24 时，没有颜色表项。

位图信息头和颜色表组成位图信息，BITMAPINFO 结构定义如下：

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;     // 位图信息头
    RGBQUAD bmiColors[1];           // 颜色表
} BITMAPINFO;
```

（5）位图数据

位图数据记录了位图的每一个像素值，记录顺序在扫描行内是从左到右,扫描行之间是从下到上。位图的一个像素值所占的字节数：

当 biBitCount=1 时，8 个像素占 1 个字节；

当 biBitCount=4 时，2 个像素占 1 个字节；

当 biBitCount=8 时，1 个像素占 1 个字节；

当 biBitCount=24 时，1 个像素占 3 个字节；

Windows 规定一个扫描行所占的字节数必须是 4 的倍数(即以 long 为单位),不足的以 0 填充，一个扫描行所占的字节数计算方法：

```
DataSizePerLine= (biWidth×biBitCount+31)/8;  // 一个扫描行所占的字节数
DataSizePerLine= DataSizePerLine/4×4;        // 字节数必须是 4 的倍数
```

位图数据的大小(不压缩情况下)：

DataSize= DataSizePerLine×biHeight;

14. 编写一个程序，它可显示 JPG 的图形文件。

**答：**

JPEG 是由 Joint Photographic ExpertsGroup(联合图像专家组规范)制定的图像压缩格式，又称 JPG 格式。它是最为常见的一种图形文件格式。JPEG 格式通过精确地记录每个像素的光亮，同时平均它们的色调，将人眼无法分辨的细节删除。可实现极高的压缩率。JPG 格式支持 24 位颜色。

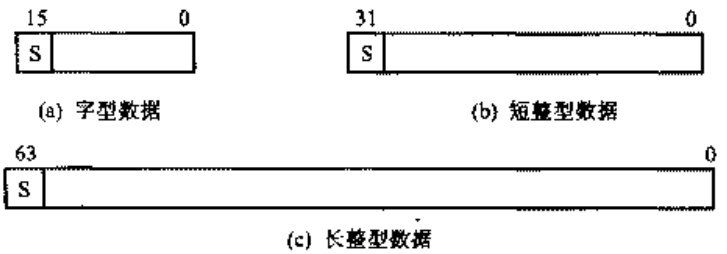
JPEG 格式比较复杂，感兴趣者可以自行查找 JPEG 格式的有关资料，再写出相应的汇编语言程序。

# 第十一章　数值运算协处理器

1．简述协处理器可以处理的3种整型数据格式、3种浮点数据格式及其表示范围。

**答：**

3种整型数据格式是：16位、32位和64位有符号整数。其结构如题图11.1所示。



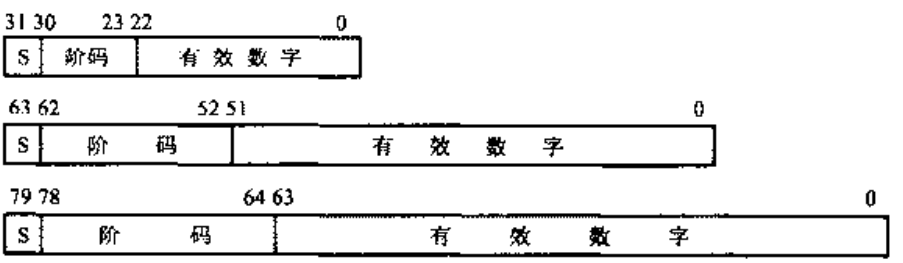

题图11.1　有符号整型数据的数据格式

（1）16位字型数据的表示范围是-32 768~32 767；

（2）32位短整型数据的表示范围是-2 147 483 648~2 147 483 647；

（3）64位长整型数据的表示范围是$-9\times10^{18}$~$9\times10^{18}$。

3种浮点数据格式是：32位、64位和80位浮点数。其结构如题图11.2所示。




题图11.2　80x87中浮点数的3种数据格式

2．简述BCD码数据在内存中所占的位数和存储的形式。

**答：**

一个BCD码数据在内存中占80位，共10个字节。其最高位字节用来表示正负号，其余9个字节，每个字节内含有两个BCD码，所以，一个BCD码数据可表示18个BCD编码。

关于BCD码的正负数，有如下规定：

（1）若最高位字节的值为 0H，则表示该 BCD 码的值为正数；

（2）若最高位字节的值为 80H，则表示该 BCD 码的值为负数。

3．在 MASM V6.11 中，如何定义 3 种不同精度的浮点数？试举例说明。

**答：**

在汇编语言中，可用 DD、DQ 和 DT 来分别说明单精度、双精度和扩展精度的浮点数。在 MASM 6.11 中可用 REAL4、REAL8 和 REAL10 来分别代替 DD、DQ 和 DT。

在定义浮点数时，要使用伪指令.8087、.287 或.387 等。

例如：

```
.387
data2    REAL4    3.345E+3        ;定义单精度浮点数
data3    REAL8    321.545         ;定义双精度浮点数
data4    REAL10   254.555         ;定义扩展精度浮点数
```

4．写出满足下列要求的变量定义语句。

（1）存储双精度数 23.44 到内存 FROG 处；

（2）存储 32 位有符号整数–123 到内存 Data2 处；

（3）存储单精度数–23.8 到内存 Data3 处。

**答：**

（1）FROG REAL8 23.44

（2）Data2 DD –123

（3）Data3 REAL4 –23.8

说明：定义浮点数时，要使用伪指令.8087、.287 或.387 等。

5．什么是+0、–0 和 NAN?

**答：**

正 0：所有的数据位都是 0；

负 0：最高位为 1，其他的数据位是 0；

NAN：非法的浮点数，阶码位全为 1，有效数字不全为 0。

6．将下列各数转换成单精度浮点数。

（1）28.75　　（2）624

（3）–0.615　　（4）+0.0

（5）–1000.5　　（6）89.6

**答：**

（1）进制转换：$(28.75)_{10}=(1\ 1100.11)_2$

规格化：$(1\ 1100.11)_2=1.1100\ 11\times10^4$

计算阶码：100+0111 1111=1000 0011

综合上述可得：$(28.75)_{10}$的单精度浮点数为：

0100 0001 1110 0110 0000 0000 0000 0000

（2）进制转换：$(624)_{10}=(1001\ 1101\ 0111\ 0000\ 1100\ 0100)_2$

规格化：$(1001\ 1101\ 0111\ 0000\ 1100\ 0100)_2=1.0011\ 1010\ 1110\ 0001\ 1000\ 100\times10^{-1}$

计算阶码：1001+0111 1111=1000 1000

综合上述可得：$(624)_{10}$的单精度浮点数为：

0100 0100 0001 1100 0000 0000 0000 0000

（3）进制转换：$(0.615)_{10}=(0.1001\ 1101\ 0111\ 0000\ 1100\ 0100)_2$

规格化：$(0.1001\ 1101\ 0111\ 0000\ 1100\ 0100)_2=1.0011\ 1010\ 1110\ 0001\ 1000\ 100\times10^{-1}$

计算阶码：1111 1111+0111 1111=0111 1110

综合上述可得：$(-0.615)_{10}$的单精度浮点数为：

1011 1111 0001 1101 0111 0000 1100 0100

（4）(以下省略计算过程)

$(+0.0)_{10}$的单精度浮点数为 0000 0000 0000 0000 0000 0000 0000 0000

（5）$(-1000.5)_{10}$的单精度浮点数为 1100 0100 0111 1100 0010 0000 0000 0000

（6）$(89.6)_{10}$的单精度浮点数为 0101 0010 1101 0011 0011 0011 0011 0011

7. 将下列单精度浮点数转换成十进制数。

（1）1100 0000 1111 0000 0000 0000 0000 0000

（2）0011 1111 0001 0000 0000 0000 0000 0000

（3）0100 0011 1001 1001 0000 0000 0000 0000

（4）0100 0000 0000 0000 0000 0000 0000 0000

（5）0100 0001 0010 0000 0000 0000 0000 0000

**答：**

（1）将单精度浮点数 1100 0000 1111 0000 0000 0000 0000 0000 分割成 3 部分，可得：符号位是 1，阶码是 1000 0001，尾数是 111 0000 0000 0000 0000 0000。

还原阶码：1000 0001-0111 1111=10

该浮点数的规格化形式：$1.111\times2^2$

该浮点数的非规格化形式：111.1

该浮点数的十进制数为：7.5。

（2）将单精度浮点数 0011 1111 0001 0000 0000 0000 0000 0000 分割成 3 部分，可得：符号位是 0，阶码是 0111 1110，尾数是 001 0000 0000 0000 0000 0000。

还原阶码：0111 1110-0111 1111=1111 1111

该浮点数的规格化形式：$1.001\times2^{-1}$

该浮点数的非规格化形式：0.1001

该浮点数的十进制数为：0.5625。

（3）将单精度浮点数 0100 0011 1001 1001 0000 0000 0000 0000 分割成 3 部分，可得：符号位是 0，阶码是 1000 0111，尾数是 001 1001 0000 0000 0000 0000。

还原阶码：1000 0111-0111 1111=1000

该浮点数的规格化形式：$1.0011001\times2^{8}$

该浮点数的非规格化形式：100110010

该浮点数的十进制数为：306.0。

（4）将单精度浮点数 0100 0000 0000 0000 0000 0000 0000 0000 分割成 3 部分，可得：符号位是 0，阶码是 1000 0000，尾数是 000 0000 0000 0000 0000 0000。

还原阶码：1000 0000-0111 1111=1

该浮点数的规格化形式：$1.00\times2^{1}$

该浮点数的非规格化形式：10

该浮点数的十进制数为：2.0。

（5）将单精度浮点数 0100 0001 0010 0000 0000 0000 0000 0000 分割成 3 部分，可得：符号位是 0，阶码是 1000 0010，尾数是 010 0000 0000 0000 0000 0000。

还原阶码：1000 0010-0111 1111=11

该浮点数的规格化形式：$1.01\times2^{3}$

该浮点数的非规格化形式：1010

该浮点数的十进制数为：10.0。

8. 状态寄存器中，$C_3$~$C_0$ 位的作用是什么？

**答：**

$C_3$~$C_0$ 是 4 位条件编码位，其组合含义如题表 11.1 所示。

**题表 11.1 状态寄存器中条件编码位的组合含义**

| 指 令 | $C_3$ | $C_2$ | $C_1$ | $C_0$ | 功 能 |
|---|---|---|---|---|---|
| FTST、FCOM | 0 | 0 | × | 0 | ST＞操作数或(0 FTST) |
| | 0 | 0 | × | 1 | ST＜操作数或(0 FTST) |
| | 1 | 0 | × | 0 | ST＝操作数或(0 FTST) |
| | 1 | 1 | × | 1 | ST 不可比较 |
| FPREM | $Q_1$ | 0 | $Q_0$ | $Q_2$ | $Q_2Q_1Q_0$ 是商的右边 3 位 |
| | ? | 1 | ? | ? | 未完成 |
| FXAM | 0 | 0 | 0 | 0 | +unnormal |
| | 0 | 0 | 0 | 1 | +NAN |
| | 0 | 0 | 1 | 0 | -unnormal |
| | 0 | 0 | 1 | 1 | -NAN |

续表

| 指　令 | $C_3$ | $C_2$ | $C_1$ | $C_0$ | 功　能 |
|---|---|---|---|---|---|
| FXAM | 0 | 1 | 0 | 0 | +normal |
| | 0 | 1 | 0 | 1 | +∞ |
| | 0 | 1 | 1 | 0 | –unnormal |
| | 0 | 1 | 1 | 1 | –∞ |
| | 1 | 0 | 0 | 0 | +0 |
| | 1 | 0 | 0 | 1 | 空 |
| | 1 | 0 | 1 | 0 | -0 |
| | 1 | 0 | 1 | 1 | 空 |
| | 1 | 1 | 0 | 0 | +denormal |
| | 1 | 1 | 0 | 1 | 空 |
| | 1 | 1 | 1 | 0 | -denormal |
| | 1 | 1 | 1 | 1 | 空 |

其中，normal —— 标准的浮点数，unnormal —— 有效数字前面是 0，如 0.XXXX，denormal——阶码是最大的负值，NAN——非法的浮点数，阶码位全为 1，有效数字不全为 0。

9. 状态寄存器中，IE 位的作用是什么？

**答：**

该状态位用来表明执行了一个错误的操作，如求负数的平方根，也可用来表明堆栈的溢出错误、不确定的格式(0/0，∞，–∞等)错误，或用 NAN 作为操作数。

10. 在 80x87 中如何选择舍入模式?

**答：**

在 80x87 中使用控制寄存器的 RC 控制位选择舍入模式。

当 RC=00 时，取最接近或偶数；

当 RC=01 时，舍入成负无穷；

当 RC=10 时，截成 0；

当 RC=11 时，舍入成正无穷。

11. 协处理器指令的一般命名规则有哪些？

**答：**

协处理器指令的操作符(或助忆符)在命名设计时，遵循了下列规则：

（1）在操作符后面加上字母 P：表示该指令执行完后，还进行一次堆栈弹出操作。如 FADD 和 FADDP 等；

（2）在操作符后面加上字母 R：表示该操作是反模式，它仅限于减法、除法指令。如 FSUB 和 FSUBR 等；

（3）操作符的第 2 个字母是 I：表示内存中数据是整数。它对加、减、乘除指令都有效。

（4）操作符的第 2 个字母是 N：表示在指令执行之前检查非屏蔽数值性错误。如 FSAVE 和 FNSAVE 等，前者称为等待形式（wait version），后者称为非等待形式（no-wait version）。

在使用.8087 伪指令情况下，汇编程序会在等待形式的指令前面加上指令 WAIT，而在非等待形式的指令前面加上空操作指令 NOP。

12. 哪条协处理器指令用到微处理器的 AX 寄存器？

**答：**

用到 AX 寄存器的协处理器指令有：

```
FSTSW     AX
FNSTSW    AX
```

13. 协处理器复位时，栈顶寄存器是什么？

**答：**

栈顶寄存器是 ST(0)。

14. 简述下列协处理器指令的功能。

（1）FSTSW AX　　（2）FINIT

（3）FADD　　（4）FSTP DATA

（5）FST DATA　　（6）F2XM1

（7）FFREE ST(2)　　（8）FSAVE

（9）FILD Data　　（10）FCHS

**答：**

（1）把状态寄存器的值传给寄存器 AX。

（2）初始化协处理器。

（3）从堆栈中弹出两个操作数，然后把计算的“和”压入堆栈。

（4）将协处理器堆栈栈顶的数据传送到目标操作数中。在进行数据传送时，系统自动根据控制寄存器中舍入控制位的设置把栈顶浮点数舍入成相应精度的数据。然后进行堆栈的弹出操作。

（5）将协处理器堆栈栈顶的数据传送到目标操作数中。在进行数据传送中时系统自动根据控制寄存器中舍入控制位的设置把栈顶浮点数舍入成相应精度的数据。堆栈不变化。

（6）求 $2^X-1$ 的值，$X$ 取自栈顶，结果也存入栈顶，$X$ 的取值范围为-1~1。

（7）释放堆栈寄存器，即使其标记为空，但其内容并没有改变。

（8）将全部机器状态存储到内存中，指令执行前不检查非屏蔽数值性错误。

（9）将整数 Data 压入堆栈中。

（10）改变栈顶数据的符号。

15. “除”和“反向除”有何不同？

**答：**

协处理器的“除”与“反向除”的区别在于它们的除数和被除数刚好掉转。

16. 简述 FTST 和 FXAM 指令之间的差异。

**答：**

FTST 的作用是将栈顶数据与 0 进行比较。

FXAM 的作用是检测栈顶数据是正数、负数，还是规格化数。

两指令均改变状态寄存器中 $C_3$~$C_0$ 的值，其含义如题表 11.2 所示。

**题表 11.2　FTST 与 FXAM 差异**

| 指　令 | $C_3$ | $C_2$ | $C_1$ | $C_0$ | 功　能 |
|---|---|---|---|---|---|
| FTST | 0 | 0 | × | 0 | ST＞操作数或(0 FTST) |
| | 0 | 0 | × | 1 | ST＜操作数或(0 FTST) |
| | 1 | 0 | × | 0 | ST＝操作数或(0 FTST) |
| | 1 | 1 | × | 1 | ST 不可比较 |
| FXAM | 0 | 0 | 0 | 0 | +unnormal |
| | 0 | 0 | 0 | 1 | +NAN |
| | 0 | 0 | 1 | 0 | -unnormal |
| | 0 | 0 | 1 | 1 | -NAN |
| | 0 | 1 | 0 | 0 | +normal |
| | 0 | 1 | 0 | 1 | +∞ |
| | 0 | 1 | 1 | 0 | -unnormal |
| | 0 | 1 | 1 | 1 | -∞ |
| | 1 | 0 | 0 | 0 | +0 |
| | 1 | 0 | 0 | 1 | 空 |
| | 1 | 0 | 1 | 0 | -0 |
| | 1 | 0 | 1 | 1 | 空 |
| | 1 | 1 | 0 | 0 | +denormal |
| | 1 | 1 | 0 | 1 | 空 |
| | 1 | 1 | 1 | 0 | -denormal |
| | 1 | 1 | 1 | 1 | 空 |

17. 简述 FSTSW 和 FNSTSW 指令之间的差异。

**答：**

“FSTSW　　Mem16”的作用是将控制寄存器的内容传送到 16 位内存中。

“FSTSW　　AX”的作用是将控制寄存器的内容传送到寄存器 AX 中。

“FNSTSW Mem16”的作用是将控制寄存器的内容传送到 16 位内存中，在传送前检查非屏蔽数值性错误。

“FNSTSW AX”的作用是将控制寄存器的内容传送到寄存器 AX 中，在传送前检查非屏蔽数值性错误。

18. FSQRT 指令执行后，必须测试处理器状态寄存器的哪个标志位？

**答：**

FSQRT 的作用是求栈顶数据的平方根。如果对负数求其平方根，则会发生错误。因此，要测试状态寄存器的标志位 IE。

19. 何时在程序中使用指令 FWAIT？

**答：**

FWAIT 的作用是使处理器处于等待状态，以便协处理器完成其操作。在 CPU 访问被协处理器影响的内存数据之前，要使用 FWAIT。

20. 分别选用一条协处理器指令，完成下列功能。

（1）将 π 压入栈顶；

（2）将 1.0 送入栈顶；

（3）将栈顶中的数加上寄存器 3 中的数；

（4）将栈顶中的数减去寄存器 2 中的数，结果存入寄存器 2 中。

**答：**

（1）FLD PI

（2）FLD 1

（3）FADD ST, ST(3)

（4）FSUBR ST, ST(2)

21. 对下列每一种功能要求，编写一个汇编语言程序实现之(假设未注明的变量都是单精度浮点类型)。

（1）Delta = sqrt($b^2$ – 4ac)；

（2）EBX＝π×EBX； ;EBX 是 32 位寄存器

（3）E = (a+b)/(c–d)；

（4）C=1/(27×π×F)；

（5）把 2~10 各整数的平方根依次存入单精度浮点数组 ROOTS；

（6）假设有 3 个都含有 100 个元素的双精度浮点数数组 D1、D2 和 D3，试依次计算：D3(i)=D2(i)×D1(i)，i=1, 2, …, 100。

**答：**

（1）

数据定义如下：

```
        _a      real4   3.0
        _b      real4   4.0
        _c      real4   5.0
        delta   real4   ?
        _four   real4   4.0
```

代码如下：

```
        fld     _four
        fld     _a
        fmulp   st(1), st
        fld     _c
        fmulp   st(1), st
        fld     _b
        fmul    st, st
        fsubrp  st(1), st
        fsqrt
        fstsw   ax
        test    ax, 1
        jz      no_error
        fldz
    no_error:
        fstp    delta
```

(2)

数据定义如下：

```
        _temp   dd  ?
```

代码如下：

```
        mov     _temp, ebx
        fldpi
        fld     _temp
        fmulp   st(1), st
        fstp    _temp
        mov     ebx, _temp
    END
```

(3)

数据定义如下：

```
        _a      real4   ?
        _b      real4   ?
        _c      real4   ?
        _d      real4   ?
```

```
        _e      real4   ?
```

代码如下：

```
        fld     _c
        fld     _d
        fsubp   st(1), st
        fld     _b
        fld     _a
        faddp   st(1), st
        fdivrp  st(1), st
        fstsw   ax
        test    ax, 80h             ;检查是否有错误
        jz      noErr
        ;…                          ;错误处理程序
    noErr:
        fstp        _e
```

(4)

数据定义如下：

```
        _p27    real4       27.0
        _F      real4       ?
        _C      real4       ?
```

代码如下：

```
        fldpi
        fld     _F
        fmulp   st(1), st
        fld     _p27
        fmulp   st(1), st
        fld 1
        fdivrp  st(1), st
        fstsw   ax
        test    ax, 80h             ;检查是否有错误
        jz      noErr
        ;…                          ;错误处理程序
    noErr:
        fstp        _C
```

(5)

```
    .model small
    .386
    .387
    .stack 200h
```

```
.data
    Source  real4   2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0
    Roots   real4   9 dup(?)
.CODE
    .startup
    mov   bx, 0
    mov   cx, 9
Next:
    fld   Source[bx]
    fsqrt
    fstp  ROOTS[bx]
    add   bx, 4
    loop  Next
    .exit
END
```

(6)

```
.model small
.386
.387
.stack 200h
.data
    D1  real8   100 dup(?)
    D2  real8   100 dup(?)
    D3  real8   100 dup(?)
.CODE
    .startup
    mov   bx, 0
    mov   cx, 100
Next:
    fld    D1[bx]
    fld    D2[bx]
    fmulp  st(1), st
    fstp   D3[bx]
    add    bx, 8
    loop   Next
    .exit
END
```

22. 根据公式 $Log_{10}X=Log_2X / Log_210$，编写子程序 LOG10，它用以求栈顶以 10 为底的对数值，结果仍存入栈顶。再按公式 $D3(i)=20\times Log_{10}(D2(i)/D1(i))$计算 20 组数据，其中：

D1、D2 和 D3 均是含有 20 个元素的单精度浮点数数组。

**答：**

```
.model small
.386
.387
.stack 200h
.data
    _temp   real4   ?
    D1      real4   20 dup(?)
    D2      real4   20 dup(?)
    D3      real4   20 dup(?)
    _p20    real4   20.0
.CODE
LOG10       proc
    fstp    _temp
    fldl
    fld     _temp
    fyl2x
    fldl2t
    fdivp   st(1), st
    ret
LOG10       endp
    .startup
    mov     bx, 0
    mov     cx, 20
Next:
    fld     D2[bx]
    fld     D1[bx]
    fdivp   st(1), st
    call    LOG10
    fld     _p20
    fmulp   st(1), st
    fstp    D3[bx]
    add     bx, 4
    loop    Next
    .exit
END
```

# 第十二章　汇编语言和C语言

1. 把教材12.3节中的C语言程序改写成汇编语言程序。

答:

```
Data1   segment
        insert      dw  0
        cap_key     dw ?
        num_key     dw  ?
        k           db  ?
        key         db  ?
        row         db  0
        col         db  0
        ch1         db 0
        ch2         db 0
        ascii_str   db 3 dup(' '), '$'
        insert_mode db  "INS$"
        caps_mode   db "CAP$"
        num_mode    db "NUM$"
        empty_mode  db  "   $"
        msg1        db "This is a simple screen editor.$"
        msg2        db "YOu can move cursor by Arrow keys, "
                    db "TAB/Shift-TAB, Home and End.$"
        msg3        db "You can press ^W for scroll up or ^Z for scroll
                    down.$"
        msg4        db "It have some functions, such as insert/modify
                    a char.$"
        msg5        db "If you press a function key, or key combined "
                    db  "with Alt, Ctrl, Shift, it will display the "
                    db  "key's scan code.$"
        msg6        db "The program exits when you press ESCAPE.$"
Data1   ends
Code1   segment
        assume  cs:Code1, ds:Data1

;滚动屏幕
```

```
;输入参数:
;ah: 滚动模式(6向上,7向下),al: 屏幕滚动的行数,0为清屏
;ch&cl: 滚动矩阵左上角坐标
;dh&dl: 滚动矩阵右下角坐标
;bh: 填充滚动矩阵底部空行所使用的文本属性
cls     proc  uses ax bx cx dx
        int   10h
        ret
cls     endp
;取当前光标的位置,将横坐标和纵坐标分别存在寄存器dh和dl中
get_cursor    proc  uses ax bx cx
        mov   ah, 3
        mov   bh, 0
        int   10h
        ret
get_cursor endp
;设置光标位置,横坐标和纵坐标分别存在寄存器dh和dl中
locate  proc  uses ax bx cx dx
        mov   ah, 2
        mov   bh, 0
        int   10h
        ret
locate  endp
;在指定位置显示字符串
;指定位置横坐标和纵坐标分别存在寄存器dh和dl中
;字符串首地址存放在寄存器ax中
disp_string   proc  uses ax bx cx dx
        mov   cx, dx
        call  get_cursor
        push  dx
        mov   dx, cx
        call  locate
        mov   dx, ax
        mov   ax, 0900h
        int   21h
        pop   dx
        call  locate
        ret
disp_string  endp
```

```
;在最后一行显示/消除大写锁定标志以及数字锁定标志
check_key_state  proc  uses ax bx cx dx
        push  dx
        mov   ah, 2
        int   16h
        push  ax
        and   al, 40h
        jz    _caps_lock_off
        mov   bx, 1
        mov   cap_key, bx
        mov   ax, offset caps_mode
        jmp   _show_caps
_caps_lock_off:
        mov   bx, 0
        mov   cap_key, bx
        mov   ax, offset empty_mode
_show_caps:
        mov   dh, 24
        mov   dl, 66
        call  disp_string
        pop   ax
        and   al, 20h
        jz    _num_lock_off
        mov   bx, 1
        mov   num_key, bx
        mov   ax, offset num_mode
        jmp   _show_num
_num_lock_off:
        mov   bx, 0
        mov   num_key, bx
        mov   ax, offset empty_mode
_show_num:
        mov   dh, 24
        mov   dl, 70
        call  disp_string
        ret
check_key_state endp
;在最后一行显示插入/修改状态标志，并改变光标形状
insert_key  proc  uses ax bx cx dx
        mov   ax, 1
```

```
        sub   ax, insert
        mov   insert, ax
        cmp   ax, 1
        jne   _InsertMode
        mov   ax, offset empty_mode
        mov   ch, 14
        jmp   _showmode
_InsertMode:
        mov   ch, 0
        mov   ax, offset insert_mode
_showmode:
        mov   dh, 24
        mov   dl, 74
        call  disp_string                    ;切换修改/插入标志
        mov   ah, 1
        mov   cl, 15
        int   10h                            ;修改光标形状
        ret
insert_key  endp
;在指定位置读字符和属性
;指定位置横坐标和纵坐标分别存放在寄存器 dh 和 dl 中
;属性存放在 ah 中，字符存放在 al 中
read_char_attr  proc  uses bx cx dx
        call  locate
        mov   ah, 8
        mov   bh, 0
        int   10h
        ret
read_char_attr  endp
;在指定位置写字符和属性
;指定位置横坐标和纵坐标分别存放在寄存器 dh 和 dl 中
;属性存放在 ah 中，字符存放在 al 中
write_char_attr  proc  uses ax bx cx dx
        call  locate
        mov   bl, ah
        mov   ah, 9
        mov   bh, 0
        mov   cx, 1
        int   10h
```

```
        ret
write_char_attr  endp
;在指定位置之后的字符和属性向右移若干个位置
;指定位置横坐标和纵坐标分别存放在寄存器 dh 和 dl 中
;移动数量存放在 cx 中
move_right  proc  uses ax bx cx dx si di
        mov   ax, dx
        mov   ah, 0
        add   ax, cx
        mov   si, ax
        mov   di, 79
_next_char:
        cmp   di, si
        jb    _move_right_over
        mov   bx, di
        sub   bx, cx
        mov   dl, bl
        call  read_char_attr
        mov   bx, di
        mov   dl, bl
        call  write_char_attr
        dec   di
        jmp   _next_char
_move_right_over:
        ret
move_right  endp
;把字符的 ASCII 码转换成字符串，字符存放在寄存器 al 中
ctos    proc  uses ax bx
        mov   bl, ' '
        mov   ascii_str, bl
        mov   ascii_str[1], bl
        mov   ascii_str[2], bl
        mov   bh, 10
        div   bh
        add   ah, '0'
        mov   ascii_str[2], ah
        mov   ah, 0
        div   bh
        cmp   ah, 0
```

```
        je    ctos_over
        add   ah, '0'
        mov   ascii_str[1], ah
        cmp   al, 0
        je    ctos_over
        add   al, '0'
        mov   ascii_str, al
ctos_over:
        ret
ctos    endp
Start:
        mov   ax, Data1
        mov   ds, ax
        mov   ah, 6
        mov   al, 0
        mov   cx, 0
        mov   dh, 24
        mov   dl, 79
        mov   bh, 7
        call  cls
        mov   dx, 0
        mov   ax, offset msg1
        call  disp_string
        mov   dx, 200h
        mov   ax, offset msg2
        call  disp_string
        mov   dx, 400h
        mov   ax, offset msg3
        call  disp_string
        mov   dx, 600h
        mov   ax, offset msg4
        call  disp_string
        mov   dx, 800h
        mov   ax, offset msg5
        call  disp_string
        mov   dx, 0b00h
        mov   ax, offset msg6
        call  disp_string
        mov   dx, 0
```

```
        call  locate
_check_ch1:
        mov   al, ch1
        cmp   al, 1bh          ;比较 ch1 是否为 escape
        jne   _not_escape
_is_escape:
        mov   ah, 6
        mov   al, 0
        mov   cx, 0
        mov   dh, 24
        mov   dl, 79
        mov   bh, 7
        call  cls
        mov   ax, 4c00h        ;程序结束
        int   21h
_not_escape:
        mov   ah, 1
        int   16h
        jnz   key_been_pressed
        call  check_key_state
        jmp   _check_ch1
key_been_pressed:
        mov   ah, 0
        int   16h
        mov   key, al
        mov   ch1, al
        mov   ch2, ah
        cmp   al, 0            ;判断 ch1 是否为 0
        je    _deal_ch2
        cmp   al, 9            ;判断是否为 TAB 键
        jne   _CTRL_w
        mov   ah, col
        and   ah, 0f8h
        add   ah, 8
        cmp   ah, 80
        jb    _save_col
        sub   ah, 80           ;ah 值不会大过 87，因此可以将除法取余数
_save_col:                     ;简化为一次减法，以减少计算量
        mov   col, ah
```

```
        jmp   _locate1
_CTRL_w:
        cmp   al, 23          ;判断是否为 CTRL+W 键
        jne   _CTRL_z
        mov   ah, 7
        mov   cx, 0
        mov   dh, 24
        mov   dl, 79
        mov   al, 1
        mov   bh, 7
        call  cls
        inc   row
        jmp   _locate1
_CTRL_z:
        cmp   al, 26          ;判断是否为 CTRL+Z 键
        jne   _default1
        mov   ah, 6
        mov   cx, 0
        mov   dh, 24
        mov   dl, 79
        mov   al, 1
        mov   bh, 7
        call  cls
        jmp   _locate1
_default1:
        cmp   al, 1bh         ;判断是否为 ESCAPE 键
        je    _is_escape
        mov   bx, insert
        cmp   bx, 0
        jne   _write_char
        mov   dh, row
        mov   dl, col
        mov   cx, 1
        call  move_right
_write_char:
        mov   dh, row
        mov   dl, col
        mov   ah, 31
        call  write_char_attr
```

```
        mov   ah, col
        inc   ah
        cmp   ah, 80
        jb    _save_col2
        sub   ah, 80          ;ah 值不会大过 80，因此可以将除法取余数
_save_col2:                   ;简化为一次减法，以减少计算量
        mov   col, ah
_locate1:
        mov   dh, row
        mov   dl, col
        call  locate
        jmp   _check_ch1
_deal_ch2:
        cmp   ah, 72          ;判断是否为 UP_ARROW 键
        jne   _down_arrow
        mov   al, row
        cmp   al, 0
        ja    _next1
        mov   al, 24
        jmp   _save_row1
_next1:
        dec   al
_save_row1:
        mov   row, al
        jmp   _locate2
_down_arrow:
        cmp   ah, 80          ;判断是否为 DOWN_ARROW 键
        jne   _left_arrow
        mov   al, row
        inc   al
        cmp   al, 25
        jb    _save_row2
        mov   al, 0
_save_row2:
        mov   row, al
        jmp   _locate2
_left_arrow:
        cmp   ah, 75          ;判断是否为 LEFT_ARROW 键
        jne   _right_arrow
```

```
        mov   al, col
        cmp   al, 0
        ja    _next2
        mov   al, 79
        jmp   _save_col3
_next2:
        dec   al
_save_col3:
        mov   col, al
        jmp   _locate2
_right_arrow:
        cmp   ah, 77              ;判断是否为 RIGHT_ARROW 键
        jne   _shift_tab
        mov   al, col
        inc   al
        cmp   al, 80
        jb    _save_col4
        mov   al, 0
_save_col4:
        mov   col, al
        jmp   _locate2
_shift_tab:
        cmp   ah, 15              ;判断是否为 SHIFT+TAB 键
        jne   _home_key
        mov   al, col
        mov   bl, al
        and   al, 0f8h
        cmp   al, bl
        jne   _save_col5
        sub   al, 8
        jge   _save_col5
        mov   al, 72
_save_col5:
        mov   col, al
        jmp   _locate2
_home_key:
        cmp   ah, 71              ;判断是否为 HOME 键
        jne   _end_key
        mov   al, 0
```

```
        mov   col, al
        jmp   _locate2
_end_key:
        cmp   ah, 79              ;判断是否为 END 键
        jne   _insert_key
        mov   al, 79
        mov   col, al
        jmp   _locate2
_insert_key:
        cmp   ah, 82              ;判断是否为 INSERT 键
        jne   _default2
        call  insert_key
        jmp   _locate2
_default2:
        mov   al, ah
        mov   ah, 0
        call  ctos
        mov   bx, insert
        cmp   bx, 0
        jne   _disp_string
        mov   dh, row
        mov   dl, col
        mov   cx, 3
        call  move_right
_disp_string:
        mov   dh, row
        mov   dl, col
        mov   ax, offset ascii_str
        call  disp_string
        mov   ah, col
        add   ah, 3
        cmp   ah, 80
        jb    _save_col6
        sub   ah, 80              ;ah 值不会大过 80，因此可以将除法取余数
_save_col6:                       ;简化为一次减法，以减少计算量
        mov   col, ah
_locate2:
        mov   dh, row
        mov   dl, col
```

```
            call  locate
            jmp   _check_ch1
Code1       ends
            end Start
```

2. 编写 C 语言程序，输出下面表达式的值，要求该表达式的计算用嵌入汇编语言程序段的方法来实现（注：题中所有变量都是整型）。

(1) 1230 + 'A' + a

(2) b×b - 4×a×c

(3) (a + b) / (c + d)

(4) 9×c / 5 + 32

(5) (a % 9 + 89) ×8

(6) x×x + y×y

**答：**

(1)

```
#include <stdio.h>
void main(void)
{
  int a;
  a = 100;
  asm{
    push ax
    push bx
    mov  ax, a
    add  a, 1230
    mov  bl, 'A'
    mov  bh, 0
    add  ax, bx
    mov  a, ax
    pop  bx
    pop  ax
  }
  a = 1230 + a + 'A';
  printf("s=%d\n", a);
}
```

(2)

```
#include <stdio.h>
void main(void)
{
```

```
    int a, b, c;
    int result;
    a = 3;
    b = 4;
    c = 5;
    asm{
      push  ax
      push  cx
      mov   ax, b
      mov   cx, ax
      mul   cx
      mov   result, ax
      mov   ax, 4
      mov   cx, a
      mul   cx
      mov   cx, c
      mul   cx
      mov   cx, result
      sub   cx, ax
      mov   result, cx
      pop   cx
      pop   ax
    }
    printf("result=%d\n", result);
  }
```

(3)

```
  #include <stdio.h>
  void main(void)
  {
    int a, b, c, d;
    int result;
    a = 30;
    b = 42;
    c = 2;
    d = 8;
    asm{
      push  ax
      push  cx
      push  dx
```

```
        mov   ax, a
        add   ax, b
        mov   cx, c
        add   cx, d
        mov   dx, 0
        div   cx
        mov   result, ax
        pop   dx
        pop   cx
        pop   ax
      }
      printf("result=%d\n", result);
    }
```

(4)

```
    #include <stdio.h>
    void main(void)
    {
      int c, result;
      c = 3;
      asm{
        push  ax
        push  cx
        pop   dx
        mov   ax, 9
        mov   cx, c
        mul   cx
        mov   cx, 5
        div   cx
        add   ax, 32
        mov   result, ax
        pop   dx
        pop   cx
        pop   ax
      }
      printf("result=%d\n", result);
    }
```

(5)

```
    #include <stdio.h>
    void main(void)
```

```
{
  int a;
  int result;
  a = 1;
  asm{
    push  ax
    push  cx
    pop   dx
    mov   dx, 0
    mov   ax, a
    mov   cx, 9
    div   cx
    add   dx, 89
    mov   ax, dx
    shl   ax, 1
    shl   ax, 1
    shl   ax, 1
    mov   result, ax
    pop   dx
    pop   cx
    pop   ax
  }
  printf("result=%d\n", result);
}
```

(6)

```
#include <stdio.h>
void main(void)
{
  int x, y;
  int result;
  x = 3;
  y = 4;
  asm{
    push  ax
    push  bx
    push  cx
    push  dx
    mov   ax, x
    mov   cx, ax
```

```
    mul   cx
    mov   bx, ax
    mov   ax, y
    mov   cx, ax
    mul   cx
    add   ax, bx
    mov   result, ax
    pop   dx
    pop   cx
    pop   bx
    pop   ax
  }
  printf("result=%d\n", result);
}
```

3. 用汇编语言编写函数 Display(Data)，其功能是在当前光标处显示无符号整数 Data，然后，编写一个 C 语言程序调用 Display 来显示整型变量的值。

**答：**

```
#include <stdio.h>

void Display(int Data)
{
  char  buffer[6];

  asm{
    push  ax
    push  bx
    push  cx
    push  dx
    push  si
    mov   byte ptr buffer[5], '$'
    mov   si, 0
    mov   cx, 5
  }
  Next1:
  asm{
    mov   byte ptr buffer[si], '' //清理buffer
    inc   si
    loop  Next1
    mov   ax, Data
```

```
    mov   cx, 10
    mov   si, 4
  }
  Next2:
  asm{
    mov   dx, 0
    div   cx
    add   dl, 30h
    mov   byte ptr buffer[si], dl
    cmp   ax, 0
    je    DispStr
    dec   si
    jmp   Next2
  }
  DispStr:
  asm{
    mov   ah, 9
    lea   dx, buffer
    int   21h
    pop   si
    pop   dx
    pop   cx
    pop   bx
    pop   ax
  };
}//end of Display

void main(void)
{
  int x;
  x = 100;
  Display(x);
}
```

4. 用汇编语言实现下列C语言标准函数，并在C语言程序中验证之(假设未指明的变量都是整型)。

(1) isalpha(int Ascii)　/*若Ascii是字母的Ascii码，则其函数值为真，否则为假*/

(2) isxdigit(int Ascii)　/*若Ascii是十六进制字符('0'~'9'、'A'~'F'和'a'~'f')，那么，其函数值为真，否则为假*/

(3) strlwr(char*s)　/*把字符串s中的字母转换成小写*/

（4）strchr(char*s1, int Ascii) /*在字符串 s1 中查找是否存在字符 Ascii。若不存在，则返回 NULL(即 0)；否则，返回指向该字符在字符串中位置的指针*/

（5）strncmp(char*s1, char*s2, int Len) /*比较字符串 s1 和 s2 前 Len 个字符，若 s1<s2，其值小于 0；s1==s2，其值为 0；否则，其值大于 0*/

（6）strncpy(char*Dest, char*Src, int Len) /*把 Src 串中前 Len 个字符复制到 Dest 中*/

（7）memset(void*Buff, int Data, int Len) /*把用 Data 填充 Buff 前 Len 个存储单元*/

**答：**

（1）函数 int isalpha(int Ascii)判断参数 Ascii 对应的是否属于大写或小写字母。如果是，则返回值非 0；如果否，则返回 0。

汇编语言源程序（文件名：ex12_4_1.asm）如下：

```
; Called from main program in C12_4_1.C
; Assemble with ML /c ex12_4_1.ASM
.MODEL  small, c
isalpha  PROTO C ascii:SWORD
.CODE
isalpha  PROC  C ascii:SWORD
    mov  ax, ascii          ; Load Arg1 into AX
    cmp  ax, 'A'
    jb   _no
    cmp  ax, 'Z'
    jbe  _yes
    cmp  ax, 'a'
    jb   _no
    cmp  ax, 'z'
    ja   _no
_yes:
    mov  ax, 1              ; Leave return value in AX
    ret
_no:
    mov  ax, 0              ; Leave return value in AX
    ret
isalpha  ENDP
    END
```

C 语言源程序（文件名：c12_4_1.c）如下：

```
/* isalpha procedure is in ex12_4_1.ASM */
/* Compile with bcc C12_4_1.C ex12_4_1.obj */
#include <stdio.h>
extern int isalpha(int ascii);
```

```
void main()
{
  if (isalpha('A')) printf( "'A' is alphabet character\n");
  else printf( "'A' is not alphabet character\n");
  if (isalpha('z')) printf( "'z' is alphabet character\n");
  else printf( "'z' is not alphabet character\n");
  if (isalpha('1')) printf( "'1' is alphabet character\n");
  else printf( "'1' is not alphabet character\n");
}
```

（2）函数 int isxdigit(int Ascii)判断参数 Ascii 对应的是否属于十六进制字符（'0' ~ '9'、'A '~' F '、' a '~' f '）。如果是，则返回值非 0；如果不是，则返回 0。

汇编语言源程序（文件名：ex12_4_2.asm）如下：

```
; Called from main program in C12_4_2.C
; Assemble with ML/c ex12_4_2.ASM
.MODEL small, c
isxdigit PROTO C ascii:SWORD
.CODE
isxdigit PROC C ascii:SWORD
    mov ax, ascii              ; Load Arg1 into AX
    cmp ax, '0'
    jb _no
    cmp ax, '9'
    jbe _yes
    cmp ax, 'A'
    jb _no
    cmp ax, 'F'
    jbe _yes
    cmp ax, 'a'
    jb _no
    cmp ax, 'f'
    ja _no
_yes:
    mov ax, 1                  ; Leave return value in AX
    ret
_no:
    mov ax, 0                  ; Leave return value in AX
    ret
isxdigit ENDP
    END
```

C 语言源程序（文件名：c12_4_2.c）如下：

```
/* isxdigit procedure is in ex12_4_2.ASM */
/* Compile with bcc C12_4_2.C ex12_4_2.obj */
#include <stdio.h>
extern int isxdigit(int ascii);
void main(){
    int     i;
    for(i=0; i<255; i++){
  if(isxdigit(i))printf("%c is a hexadecimal character\n", i);
  else printf("%c is not a hexadecimal character\n", i);
    }
}
```

（3）函数 char *strlwr(char *s)的作用是把字符串 s 中的字母转换成小写。

汇编语言源程序（文件名：ex12_4_3.asm）如下：

```
; Called from main program in C12_4_3.C
; Assemble with ML/c ex12_4_3.ASM
.MODEL  small, c

strlwr  PROTO C string:SWORD
.CODE
strlwr  PROC  C string:SWORD
    mov ax, WORD ptr string
    mov bx, ax
_next:
    mov ch, byte ptr [bx]
    cmp ch, 0
    je  _over
    cmp ch, 'A'
    jb  _no
    cmp ch, 'Z'
    ja  _no
    add ch, 20h
    mov byte ptr [bx], ch
_no:
    inc bx
    jmp _next
_over:
    ret                 ; Result returned as AX
strlwr  ENDP
```

```
    END
```

C 语言源程序（文件名：c12_4_3.c）如下：

```
/* strlwr procedure is in ex12_4_3.ASM */
/* Compile with bcc C12_4_3.C ex12_4_3.obj */

#include <stdio.h>
//#include <string.h>

extern char *strlwr(char *s);
void main(void){
    char    *String="HEAL THE WORLD, MAKE IT A BETTER PLACE.";

    printf("string prior to strlwr:%s\n", String);
    printf("string after strlwr:%s\n", strlwr(String));
}
```

（4）函数 char *strchr(char *s1, int Ascii)的作用是在字符串 s1 中寻找字符 Ascii，如果找不到则返回 NULL；否则，返回该指针在字符串中位置的指针。

汇编语言源程序（文件名：ex12_4_4.asm）如下：

```
;  Called from main program in C12_4_4.C
;  Assemble with ML/c ex12_4_4.ASM
.MODEL  small, c
strchr  PROTO C s1:SWORD, c1:BYTE
.CODE
strchr  PROC  C s1:SWORD, c1:BYTE
    mov bx, WORD ptr s1
    mov ax, 0
    mov cl, c1
_next:
    mov ch, byte ptr [bx]
    cmp ch, 0
    je  _over
    cmp ch, cl
    jne _no
    mov ax, bx
_no:
    inc bx
    jmp _next
_over:
    ret                 ; Result returned as AX
```

```
strchr  ENDP
    END
```

C 语言源程序（文件名：c12_4_4.c）如下：

```
/* strchr procedure is in ex12_4_4.ASM */
/* Compile with bcc C12_4_4.C ex12_4_4.obj */
#include <stdio.h>
//#include <string.h>
extern char *strchr(char *s1, char c1);
void main(void){
    char  *String="HEAL THE WORLD, MAKE IT A BETTER PLACE.";
    char  *ptr, c;

    c = 'M';
    printf("Source string is:%s\n", String);
    ptr = strchr(String, c);
    if(ptr)printf("The character %c is at position: %d.\n", c, ptr-
    String);
    else printf("The character was not found.\n");
}
```

（5）函数 int strncmp(char *s1, char *s2, int len)的作用是比较字符串 s1 和字符串 s2 前 len 个字符。若 s1<s2，其值小于 0；若 s1==s2，其值为 0；若 s1>s2，其值大于 0。

汇编语言源程序（文件名：ex12_4_5.asm）如下：

```
;  Called from main program in C12_4_5.C
;  Assemble with ML/c ex12_4_5.ASM
.MODEL  small, c
strncmp  PROTO C s1:SWORD, s2:SWORD, len:SWORD
.CODE
strncmp  PROC  C s1:SWORD, s2:SWORD, len:SWORD
    mov si, WORD ptr s1
    mov di, WORD ptr s2
    mov cx, len
_next:
    mov dh, byte ptr [si]
    mov dl, byte ptr [di]
    inc si
    inc di
    cmp dh, dl
    ja  _above
    jb  _below
```

```
    loop_next
    mov ax, 0
    ret                 ; Result returned as AX
_above:
    mov ax, 1
    ret                 ; Result returned as AX
_below:
    mov ax, -1
    ret                 ; Result returned as AX
strncmp  ENDP
    END
```

C语言源程序（文件名：c12_4_5.c）如下：

```
/* strncmp procedure is in ex12_4_5.ASM  */
/* Compile with bcc C12_4_5.C ex12_4_5.obj */
#include <stdio.h>
//#include <string.h>
extern int strncmp(char *s1, char *s2, int len);
void main(void){
    char    *String1= "aaabbb";
    char    *String2= "bbbccc";
    char    *String3= "ccc";
    char    *String4= "aaabbb";
    int     ptr;

    ptr=strncmp(String4, String1, 6);
    if(ptr>0)printf("String 4 is greater than String 1.\n");
    else{
        if(ptr==0) printf("String 4 is same as String 1.\n");
        else printf("String 4 is less than String 1.\n");
    }
    ptr=strncmp(String2, String1, 6);
    if(ptr>0)printf("String 2 is greater than String 1.\n");
    else{
        if (ptr==0) printf("String 1 is same as String 2.\n");
        else printf("String 2 is less than String 1.\n");
    }
    ptr=strncmp(String2, String3, 6);
    if(ptr>0)printf("String 2 is greater than String 3.\n");
    else{
```

```
            if(ptr==0)printf("String 2 is same as String 3.\n");
            else printf("String 2 is less than String 3.\n");
        }
    }
```

（6）函数 char *strncpy(char *Dest, char *Src, int len)的作用是把字符串 Src 的前 len 个字符复制到字符串 Dest 中，返回值是 Dest 的地址。

汇编语言源程序（文件名：ex12_4_6.asm）如下：

```
; Called from main program in C12_4_6.C
; Assemble with ML/c ex12_4_6.ASM
.MODEL small, c
strncpy PROTO C Dest:SWORD, Src:SWORD, len:SWORD
.CODE
strncpy PROC C Dest:SWORD, Src:SWORD, len:SWORD
    mov si, WORD ptr Src
    mov di, WORD ptr Dest
    mov cx, len
_next:
    mov ah, byte ptr[si]
    mov byte ptr[di], ah
    inc si
    inc di
    loop _next
    mov ax, WORD ptr Dest
    ret             ; Result returned as AX
strncpy ENDP
    END
```

C 语言源程序（文件名：c12_4_6.c）如下：

```
/* strncmp procedure is in ex12_4_6.ASM */
/* Compile with bcc C12_4_6.C ex12_4_6.obj */
#include <stdio.h>
//#include <string.h>
extern int strncpy(char *Dest, char *Src, int len);
void main(void){
    char        String[10];
    char        *s = "abcdefgi";
    char        *ptr;
    ptr = strncpy(String, s, 4);
    String[4] = '\0';
    printf("%s.\n", String);
```

```
}
```

（7）函数 void *memset(void *Buff, int Data, int len)的作用是用 Data 填充 Buff 的前 len 个存储单元。

汇编语言源程序（文件名：ex12_4_7.asm）如下：

```
; Called from main program in C12_4_7.C
; Assemble with ML/c ex12_4_7.ASM
.MODEL  small, c
memset  PROTO C Buff:SWORD, Data:SWORD, len:SWORD
.CODE
memset  PROC C Buff:SWORD, Data:SWORD, len:SWORD
    mov  si, WORD ptr Buff
    mov  ax, WORD ptr Data
    mov  cx, len
_next:
    mov  byte ptr [si], al
    inc  si
    loop _next
    mov  ax, WORD ptr Buff
    ret                  ; Result returned as AX
memset  ENDP
    END
```

C 语言源程序（文件名：c12_4_7.c）如下：

```
/* strncmp procedure is in ex12_4_7.ASM */
/* Compile with bcc C12_4_7.C ex12_4_7.obj */
#include <stdio.h>
//#include <string.h>
//#include <mem.h>
extern void *memset(char *Buff, int Data, int len);
void main(void){
  char buffer[] = "Hello world\n";
  printf("Buffer before memset: %s\n", buffer);
  memset(buffer, '*', 11);
  printf("Buffer after memset:  %s\n", buffer);
}
```

5. 编写一个 C 语言程序，用 TCC/BCC 命令生成汇编语言程序，分析 C 语言语句和汇编语言语句之间的实现关系。

**答：**

编写一个简单 C 语言程序（文件名：hello.c），该程序输出“Hello World!”。然后用 bcc -S hello.c 将其转换成汇编语言文件（文件名 hello.asm）。

C 语言源文件如下：

```
#include <stdio.h>
void main()
{
  printf("Hello, world!\n");
}
```

转换的汇编语言程序如下：

```
ifndef  ??version
?debug  macro
        endm
publicdll macro name
    public  name
    endm
$comm   macro   name,dist,size,count
        comm    dist name:BYTE:count*size
        endm
else
$comm   macro   name,dist,size,count
        comm    dist name[size]:BYTE:count
        endm
endif
?debug  V 300h
?debug  S "hello.c"
?debug  C E95D585F2F0768656C6C6F2E63
?debug  C E94019CA1815433A5C42435C494E434C5544455C737464696F2E68
?debug  C E94019CA1815433A5C42435C494E434C5544455C5F646566732E68
?debug  C E94019CA1816433A5C42435C494E434C5544455C5F6E66696C652E+
?debug  C 68
?debug  C E94019CA1815433A5C42435C494E434C5544455C5F6E756C6C2E68
_TEXT   segment byte public 'CODE'
_TEXT   ends
DGROUP  group   _DATA,_BSS
    assume  cs:_TEXT,ds:DGROUP
_DATA   segment word public 'DATA'
d@ label byte
d@wlabel word
_DATA   ends
```

```
    _BSS    segment word public 'BSS'
b@ label byte
b@wlabel word
_BSS    ends
_TEXT   segment byte public 'CODE'
      ;
      ; void main()
      ;
        assume  cs:_TEXT
    _main   proc near
        push bp
        mov bp,sp
      ;
      ; {
      ;  printf( "Hello, world!\n");
      ;
        mov     ax,offset DGROUP:s@
        push    ax
        call    near ptr_printf
        pop     cx
      ;
      ; }
      ;
        pop bp
        ret
    _main   endp
        ?debug  C E9
        ?debug  C FA00000000
    _TEXT   ends
    _DATA   segment word public 'DATA'
    s@  label byte
        db  'Hello, world!'
        db  10
        db  0
    _DATA   ends
    _TEXT   segment byte public 'CODE'
    _TEXT   ends
        public  _main
        extrn_printf:near
```

```
_s@ equ s@
    end
```

从转换的汇编语言程序可以看出，C 语言编译器是通过堆栈方式传递参数的。在使用参数时需要使用 BP 堆栈寄存器，因此必须在使用前保护 BP 堆栈寄存器的值并在使用后恢复。

6. 编写一个 C 语言程序，求出 2~100 之内的所有素数(大于 1，且只能被 1 和自身整除的数，称为素数)，然后把它改写汇编语言程序，并比较二者代码的。

答：

C 语言程序是：

```
#include <string.h>
#include <stdio.h>
void main(void)
{
   int   buffer[100];
   int   i, j;
   for(i=1; i<=100; ++i)buffer[i-1]=1;
   j=2;
   do{
        if(buffer[j-1])for(i=2*j;i<=100;i+=j)buffer[i-1]=0;
        j++;
   }while(j<=100);
   for(i=2; i<=100; i++)
        if(buffer[i-1])printf("%i\n", i);
}
```

相应的汇编语言程序是：

```
Data1   Segment
    IntToStrbuffer  db 5 dup(?), 13, 10, '$'
    buffer          db 101 dup(1)
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;将无符号字转换成十进制无符号形式的字符串，存放在 IntToStrbuffer 中
;Input: AX—Data
;Output:IntToStrbuffer—转化后的十进制无符号形式的字符串
IntToStr    proc uses ax bx cx dx
        mov  bx, 0
        mov  cx, 5
Next1:
```

```
        mov  IntToStrbuffer[bx], ' '          ;清理 IntToStrbuffer
        inc  bx
        loop Next1
        mov  cx, 10
        mov  bx, 4
Next2:
        mov  dx, 0
        div  cx
        add  dl, 30h
        mov  IntToStrbuffer[bx], dl
        cmp  ax, 0
        je   IntToStr_Return
        dec  bx
        jmp  Next2
IntToStr_Return:
        ret
IntToStr    endp
Start:
        mov  ax, Data1
        mov  ds, ax
        mov  bx, 1
NextInt:
        inc  bx
        cmp  bx, 100
        ja   OutputData
        mov  al, buffer[bx-1]
        cmp  al, 0
        je   NextInt
        mov  ah, 0
        mov  si, bx
MarkNext:
        add  si, bx
        cmp  si, 100
        ja   NextInt
        mov  buffer[si-1], ah
        jmp  MarkNext
OutputData:
        mov  bx, 1
```

```
FindNext:
        inc  bx
        cmp  bx, 100
        ja   finish
        mov  al, buffer[bx-1]
        cmp  al, 1
        jne  FindNext
        mov  ax, bx
        call IntToStr
        mov  ah, 9
        mov  dx, offset IntToStrbuffer
        int  21h
        jmp  FindNext
finish:
        mov  ax, 4c00h
        int  21h
Code1   Ends
    End Start
```

汇编语言程序和C语言程序的程序流程完全相同。

从源程序来看，汇编语言程序的可读性比C语言程序要差得多，而且汇编语言源程序的代码量也比C语言源程序的代码量要大得多。

但从最后生成的可执行文件来看，汇编语言生成的可执行文件为748个字节，而C语言程序生成的可执行文件为6708个字节，相比起来前者生成的可执行文件要小得多。

7. 编写一个C语言程序，求出2~999之内的所有能被9整除，且含有5的数，然后把它改写汇编语言程序。

**答：**

C语言程序是：

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
  int  buffer[18];
  int  i, j;
  for(i=2;i<1000; i++){
    if(i % 9)continue;
    itoa(i, buffer,10);
    if(!strchr(buffer,'5'))continue;
```

```
        printf("%s   ", buffer);
      }
    }
```

相应的汇编语言程序是：

```
Data1   Segment
    IntToStrbuffer  db 5 dup(?), 13, 10, '$'
Data1   Ends
Code1   Segment
    assume  cs:Code1, ds:Data1
;将无符号字转换成十进制无符号形式的字符串，存放在 IntToStrbuffer 中
;Input: AX—Data
;Output:  IntToStrbuffer——转化后的十进制无符号形式的字符串
IntToStr    proc uses ax bx cx dx
        mov  bx, 0
        mov  cx, 5
Next1:
        mov  IntToStrbuffer[bx], ' '            ;清理 IntToStrbuffer
        inc  bx
        loop Next1
        mov  cx, 10
        mov  bx, 4
Next2:
        mov  dx, 0
        div  cx
        add  dl, 30h
        mov  IntToStrbuffer[bx], dl
        cmp  ax, 0
        je   IntToStr_Return
        dec  bx
        jmp  Next2
IntToStr_Return:
        ret
IntToStr    endp
Start:
        mov  ax, Data1
        mov  ds, ax
        mov  bx, 1
NextInt:
        inc  bx
```

```
        cmp  bx, 999
        ja   finish
        mov  ax, bx
        mov  cl, 9
        div  cl
        cmp  ah, 0
        jne  NextInt
        mov  ax, bx
        call IntToStr
        mov  si, 0
NextChar:
        cmp  si, 5
        je   NextInt
        mov  dl, IntToStrbuffer[si]
        cmp  dl, '5'
        je   Output
        inc  si
        jmp  NextChar
Output:
        mov  ah, 9
        mov  dx, offset IntToStrbuffer
        int  21h
        jmp  NextInt
finish:
        mov  ax, 4c00h
        int  21h
Code1   Ends
    End Start
```

8. 用汇编语言编写一个过程 Display(Data)，其功能为在当前光标处显示无符号整数 Data，然后编写 C 语言程序调用之，以达到显示数据的作用。

**答：**

汇编语言源程序（文件名：ex12_8.asm）如下：

```
; Called from main program in C12_8.C
; Assemble with ML/c ex12_8.ASM
.MODEL  small, c
Display  PROTO C _int:SWORD
.Data
    IntToStrbuffer  db 5 dup(?), '$'
.CODE
```

```
;将无符号字转换成十进制无符号形式的字符串，存放在 IntToStrbuffer 中
;Input Parameters: AX—Data
;Output Parameters: IntToStrbuffer—转化后的十进制无符号形式的字符串
IntToStr    proc  uses ax bx cx dx
        mov  bx, 0
        mov  cx, 5
Next1:
        mov  IntToStrbuffer[bx], ' '            ;清理 IntToStrbuffer
        inc  bx
        loop Next1
        mov  cx, 10
        mov  bx, 4
Next2:
        mov  dx, 0
        div  cx
        add  dl, 30h
        mov  IntToStrbuffer[bx], dl
        cmp  ax, 0
        je   IntToStr_Return
        dec  bx
        jmp  Next2
IntToStr_Return:
        ret
IntToStr    endp

Display  PROC  C_int:SWORD
    mov  ax, @Data
    mov  ds, ax
    mov  ax, _int     ; Load Arg1 into AX
    call IntToStr
    lea  bx, IntToStrbuffer
_nextchar:
    mov  al, BYTE ptr[bx]
    cmp  al, ' '
    jnz  _output
    inc  bx
    jmp  _nextchar
_output:
    mov  si, bx
```

```
        mov  ah, 3
        mov  bh, 0
        int  10h                ;读取光标位置
        mov  ah, 13h
        mov  al, 1
        mov  bh, 0
        mov  bl, 7
        lea  cx, IntToStrbuffer
        add  cx, 5
        sub  cx, si
        push ds
        pop  es
        mov  bp, si
        int  10h                ;在光标位置显示字符串
        ret
    Display  ENDP
        END
```

C语言源程序（文件名：c12_8.c）如下：

```
/* isalpha procedure is in ex12_8.ASM  */
/* Compile with bcc C12_8.C ex12_8.obj */
#include <stdio.h>
extern void Display(int Data);
void main()
{
  printf("First data:");
     Display(12345);
     printf(",Second data:");
     Display(1234);
     printf("\nThird data:");
     Display(123);
     printf(",Fourth data:");
     Display(12);
     printf(",Fifth data:");
     Display(1);
}
```

新世纪高等学校教材



ISBN 7-04-014237-6

9 787040 142372 >

定价 19.50 元